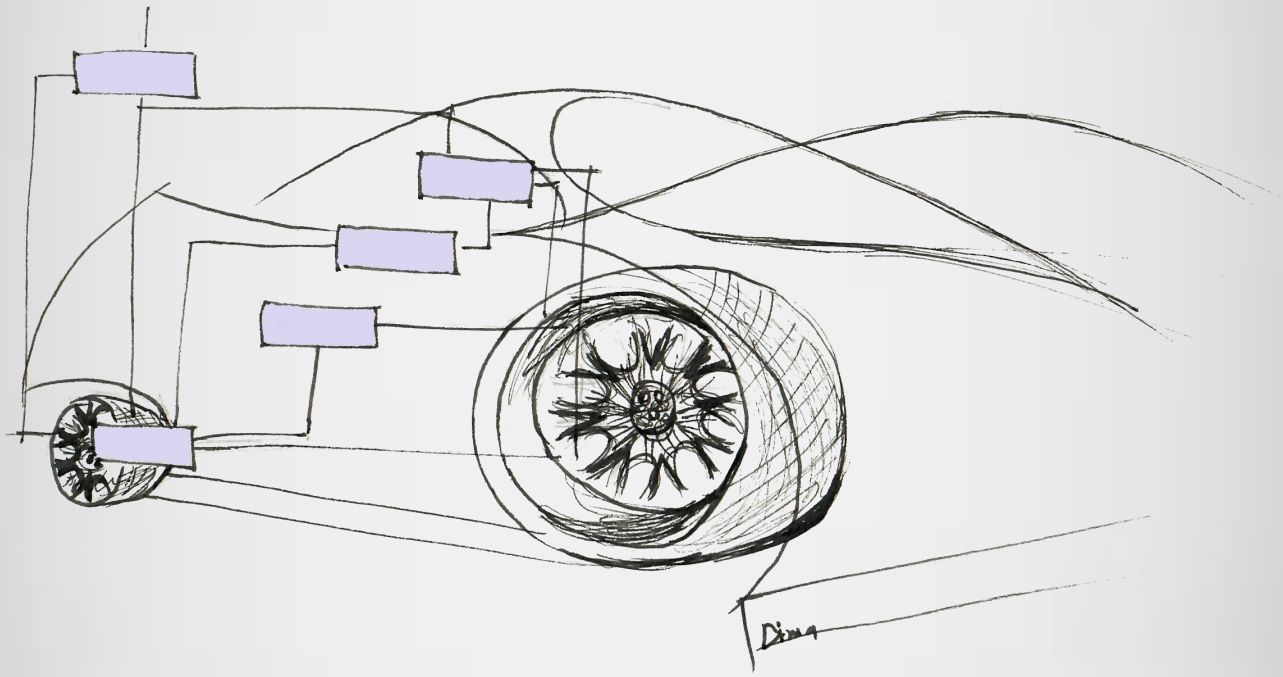
A Multilayer Secure Framework for Vehicular Systems



Mohammad Hamad

A Multilayer Secure Framework for Vehicular Systems

Mohammad Hamad

A Multilayer Secure Framework for Vehicular Systems

Von der Fakultät für Elektrotechnik, Informationstechnik, Physik

der Technischen Universität Carolo-Wilhelmina zu Braunschweig

zur Erlangung des Grades eines Doktors

der Ingenieurwissenschaften (Dr.-Ing.)

genehmigte Dissertation

von Mohammad Hamad

aus Aleppo, Syrien

eingereicht am: 25.10.2019

mündliche Prüfung am: 13.02.2020

1. Referent: Prof. Dr. Vassilis Prevelakis
2. Referent: Prof. Dr. Angelos D. Keromytis
3. Referent: Apl. Prof. Dr.-Ing. Wael Adi (Vorsitzender)

Druckjahr: 2020

"In science the credit goes to the man who convinces the world, not to the man to whom the idea first occurs."

William Osler

"The freedom to make my own mistakes was All I ever wanted."

Mance Rayder (GOT)

Acknowledgments

Creating this Ph.D. thesis is not my successful individual story; many persons had a substantial hidden effort behind it. Here is my chance to show my gratitude to all of them.

Firstly, I am very grateful to my supervisor, Professor Vassilis Prevelakis, who has guided and encouraged me to be professional. He has given me the freedom I need to choose the ideas I want to research. Without his persistent help, the work presented in this thesis would not have been possible. I am very fortunate to have had such a good supervisor!

Besides my advisor, I would like to thank Prof. Angelos Keromytis (my external supervisor) for his time and effort to read my work and for Prof. Wael Adi for chairing my Ph.D. committee.

I take this opportunity to express gratitude to my friends at the Institute of Computer and Network Engineering at TU Braunschweig for their help and support, especially, Marinos Tsantekidis, Zain A. H. Hammadeh and Saleh Mulhem.

I also thank my parents, my brothers, my sisters, and my wife's family for the unceasing encouragement, love, support, and attention. Words can not express my love and gratitude to all of them.

Last and not least, I would like to thank my beloved wife, Dima, for her sacrifices to give me the comfortable conditions to do this work. I want to express my gratitude to her for love and support, and for being in my life. Also, I want to thank her for designing the fantastic cover of my thesis. A big thank goes to my small angel, Celia, who was born at the same time I started writing this work. She gave me enormous power and motivation to proceed.

Finally, I want to thank you for reading this work.

Abstract

In recent years, significant developments were introduced within the vehicular domain, evolving the vehicles to become a network of many embedded systems distributed throughout the car, known as Electronic Control Units (ECUs). Each one of these ECUs runs a number of software components that collaborate with each other to perform various vehicle functions. Modern vehicles are also equipped with wireless communication technologies, such as WiFi, Bluetooth, and so on, giving them the capability to interact with other vehicles and roadside infrastructure. While these improvements have increased the safety of the automotive system, they have vastly expanded the attack surface of the vehicle and opened the door for new potential security risks. The situation is made worse by a lack of security mechanisms in the vehicular system which allows the escalation of a compromise in one of the non-critical sub-systems to threaten the safety of the entire vehicle and its passengers.

This dissertation focuses on providing a comprehensive framework that ensures the security of the vehicular system during its whole life-cycle. This framework aims to **prevent** the cyber-attacks against different components by ensuring secure communications among them. Furthermore, it aims to **detect** attacks which were not prevented successfully, and finally, to **respond** to these attacks properly to ensure a high degree of safety and stability of the system.

The thesis starts by developing a hybrid threat model which combines multiple existing threat modeling approaches to define more comprehensive one. This model defines (1) the various potential groups of attackers, which may threaten the vehicular system and their capabilities, (2) the potential targets (i.e., assets) of these groups and the various vulnerabilities that they include, and (3) the security requirements for these targets which should be considered to prevent the attacker from compromising them.

After defining the security requirements by using the proposed threat model, the thesis addresses the challenges of developing the security policy, which implements these requirements. The thesis presents a methodology supporting the gradual definition of the security policy. Under our methodology, the designer of each software component is responsible for formulating the security policy of their components. As components get integrated into larger sub-systems, the individual policies are merged into the subsystem policy. This continues as we go up the ladder of bigger sub-systems until we have a complete vehicle.

The thesis also shows how to enforce the developed security policy in an

efficient manner by using a lightweight distributed access framework implemented within each single ECU. The enforcement takes place at the network level, enforcing communications only between authorized components while employing data integrity mechanisms in the communication between components, even if they run on different ECUs. In this way, we provide a level of compartmentalization in the in-vehicle network. With this precondition, a malicious application might remain able to emit (a) malicious packet(s) to its remote peer(s), if it is authorized. But, at the same time, this application can be prevented from attacking other components, which it is not authorized to communicate with.

A heavy-handed security policy may adversely impact availability. Taken to the extreme, a secure system is a silent system that does not interact with its environment, and this is clearly not the intent of a security policy aimed at a vehicular platform. So we face the conundrum of increased security leading to false positives affecting availability and overall performance, against a more permissive system that may fail to detect attacks (false negatives), leading to the demise of the platform. The thesis addresses this issue by using the Red-Zone principle, whereby a tighter inner security envelope alerts the security system of a potential compromise before an actual security violation occurs. In this way, we can observe the suspect component as it operates within the Red-Zone, and characterize the event. We leverage the Red-zone principle in order to develop a run-time mechanism to detect the incidence of an attack and to prevent the attackers from gaining a foothold. The thesis defines temporal specifications for each hard real-time software component within the vehicle to be used as a baseline to define its nominal behavior. Attacks such as code injection, or Denial of Service (DoS) will usually cause a breach of this temporal specification, and thus will be detected.

Once a software component is found to have violated its security boundaries, the system needs to take some remedial action. The type of response, e.g., taking the component offline, restarting the component, initiating containment measures (e.g., resetting the entire ECU), and so on, are the responsibility of the Intrusion Response System (IRS). This thesis uses the Red-Zone principle as the basis for developing an IRS framework to manage the interaction between security and safety of the system.

KURZFASSUNG

In den letzten Jahren wurden bedeutende Entwicklungen im Bereich der Fahrzeuge vorgestellt, die die Fahrzeuge zu einem Netzwerk mit vielen im gesamten Fahrzeug verteile integrierte Systeme weiterentwickelten, den sogenannten Steuergeräten (ECU, englisch = Electronic Control Units). Jedes dieser Steuergeräte betreibt eine Reihe von Softwarekomponenten, die bei der Ausführung verschiedener Fahrzeugfunktionen zusammenarbeiten. Moderne Fahrzeuge sind auch mit drahtlosen Kommunikationstechnologien wie WiFi, Bluetooth usw. ausgestattet, die ihnen die Möglichkeit geben, mit anderen Fahrzeugen und der straßenseitigen Infrastruktur zu interagieren. Während diese Verbesserungen die Sicherheit des Fahrzeugsystems erhöht haben, haben sie die Angriffsfläche des Fahrzeugs erheblich vergrößert und die Tür für neue potenzielle Sicherheitsrisiken geöffnet. Die Situation wird durch einen Mangel an Sicherheitsmechanismen im Fahrzeugsystem verschärft, die es ermöglichen, dass ein Kompromiss in einem der unkritischen Subsysteme die Sicherheit des gesamten Fahrzeugs und seiner Insassen gefährdet kann.

Diese Dissertation konzentriert sich auf die Entwicklung eines umfassenden Rahmens, der die Sicherheit des Fahrzeugsystems während seines gesamten Lebenszyklus gewährleistet. Dieser Rahmen zielt darauf ab, die Cyber-Angriffe gegen verschiedene Komponenten zu verhindern, indem eine sichere Kommunikation zwischen ihnen gewährleistet wird. Darüber hinaus zielt es darauf ab, Angriffe zu erkennen, die nicht erfolgreich verhindert wurden, und schließlich auf diese Angriffe angemessen zu reagieren, um ein hohes Maß an Sicherheit und Stabilität des Systems zu gewährleisten.

Die Arbeit beginnt mit der Entwicklung eines hybriden Bedrohungsmodells, das mehrere bestehende Ansätze zur Bedrohungsmodellierung kombiniert, um ein umfassenderes zu definieren. Dieses Modell definiert (1) die verschiedenen potenziellen Gruppen von Angreifern, die das Fahrzeugsystem und ihre Fähigkeiten gefährden können, (2) die potenziellen Ziele dieser Gruppen und die verschiedenen Schwachstellen, die sie beinhalten und (3) die Sicherheitsanforderungen für diese Ziele, die so zu betrachten sind, dass der Angreifer sie nicht gefährden kann.

Nach der Definition der Sicherheitsanforderungen durch die Verwendung des vorgeschlagenen Bedrohungsmodells, befasst sich die Dissertation mit den Herausforderungen der Entwicklung der Sicherheitspolitik, die diese Anforderungen umsetzt. Zudem stellt sie eine Methodik zur Unterstützung der schrittweisen Definition der Sicherheitspolitik. Nach unserer Methodik ist der Designer jeder Softwarekomponente für die Formulierung

der Sicherheitspolitik ihrer Komponenten verantwortlich. Wenn Komponenten in größere Subsysteme integriert werden, werden die einzelnen Richtlinien in die Subsystemrichtlinie integriert. Dies geht noch weiter, wenn wir die Leiter größerer Subsysteme hinaufgehen, bis wir ein komplettes Fahrzeug haben.

Die Arbeit zeigt auch, wie die entwickelte Sicherheitsrichtlinie effizient durchgesetzt werden kann, indem ein leichtgewichtige Struktur für den verteilten Zugriff verwendet wird, das in jedem einzelnen ECU implementiert ist. Die Durchsetzung erfolgt auf Netzwerkebene, wobei die Kommunikation nur zwischen autorisierten Komponenten erzwungen wird, während bei der Kommunikation zwischen Komponenten Datenintegritätsmechanismen eingesetzt werden, auch wenn sie auf verschiedenen Steuergeräten laufen. Auf diese Weise sorgen wir für einen Grad der Abschottung im Fahrzeugnetz. Unter dieser Voraussetzung kann eine bösartige Anwendung in der Lage bleiben, (ein) bösartiges Paket(e) an ihre entfernten Peer(en) zu senden, wenn sie berechtigt ist. Gleichzeitig kann jedoch verhindert werden, dass diese Anwendung andere Komponenten angreift, mit denen sie nicht kommunizieren darf.

Eine schwerfällige Sicherheitsrichtlinie kann die Verfügbarkeit beeinträchtigen. Im Extremfall ist ein sicheres System ein leises System, das nicht mit seiner Umgebung interagiert, und das ist eindeutig nicht die Absicht einer Sicherheitspolitik, die auf eine Fahrzeugplattform abzielt. So stehen wir vor dem Rätsel der erhöhten Sicherheit, die zu Fehlalarmen führt, die sich auf die Verfügbarkeit und die Gesamtperformance auswirken, gegenüber einem freizügigeren System, das Angriffe (Fehlalarme) nicht erkennen kann, was zum Untergang der Plattform führt. Die Dissertation befasst sich mit diesem Thema unter Verwendung des Red-Zone-Prinzips, bei dem ein engerer innerer Sicherheitsbereich das Sicherheitssystem auf einen möglichen Kompromiss aufmerksam macht, bevor eine tatsächliche Sicherheitsverletzung eintritt. Auf diese Weise können wir die verdächtige Komponente beobachten, wie sie innerhalb der Red Zone arbeitet, und das Ereignis charakterisieren. Wir nutzen das Red-Zone-Prinzip, um einen Laufzeitmechanismus zu entwickeln, der die Häufigkeit eines Angriffs erkennt und verhindert, dass die Angreifer Fuß fassen. Die Arbeit definiert zeitliche Spezifikationen für jede Hardware-Echtzeit-Softwarekomponente innerhalb des Fahrzeugs, die als Basislinie zur Definition des nominalen Verhaltens verwendet werden soll. Angriffe wie Code Injection oder Denial of Service (DoS) führen in der Regel zu einer Verletzung dieser zeitlichen Spezifikation und werden so erkannt.

Sobald festgestellt wird, dass eine Softwarekomponente ihre Sicherheitsgrenzen überschritten hat, muss das System einige Abhilfemaßnahmen ergreifen. Die Art der Reaktion, z.B. das Offline-Schalten der Komponente, das Neustarten der Komponente, das Einleiten von Sicherheitsmaßnahmen (z.B. das Zurücksetzen des gesamten Steuergeräts) usw., liegt in der Verantwortung des Intrusion Response System (IRS). Diese Arbeit verwendet das Red-Zone-Prinzip als Grundlage für die Entwicklung eines IRS-Strukturs zur Steuerung der Interaktion zwischen Sicherheit und Schutz des Systems.

Contents

Part I

THE LANDSCAPE

"IoV without security == Internet of Vulnerabilities"

1 Introduction

1.1 Motivation

In recent years, vehicles manufacturing has changed significantly: vehicles moved from a largely electro-mechanical system to an Electrical and Electronic (E/E) system. This can be seen in the increased use of automotive embedded systems and the large quantity of embedded software which is integrated within every single vehicle [Bro+07; Cha09], as well as the high percentage of the production cost attributable to developing these integrated embedded systems and automotive software [Bro06a].

Continuous improvements to vehicle manufacturing have made developing Intelligence Transportation Systems (ITS) achievable. Many technologies have been integrated within modern vehicles to give them the capability to interact with the outside world. Figure 1.1 shows that in Germany by 2023, more than 1.1 million new cars will be equipped with infotainment and communication systems to support communications with other vehicles and infrastructure on the road. Ensuring road safety is the main motivation behind introducing such technologies. Most road accidents are caused by human errors (e.g., 70% of road accidents in Germany in 2015 were a result of human mistakes [Dek17]), thus, using safety software applications within vehicles aims to decrease the number of such accidents and therefore the number of fatalities. In addition, improving road usability as well as drivers' and passengers' comfort are other benefits of such technologies.

The revolutionary changes do not stop here; the move toward full autonomous vehicle technology is the next goal of all car makers. However, there are many problems which need to be addressed before we will be able to see the benefits of connected and autonomous vehicles in reality [DC+14; Lia+15]. Cybersecurity is one of the most serious and urgent of these issues. Understanding and mitigating the security issues within each vehicle is the first step towards secure ITS.

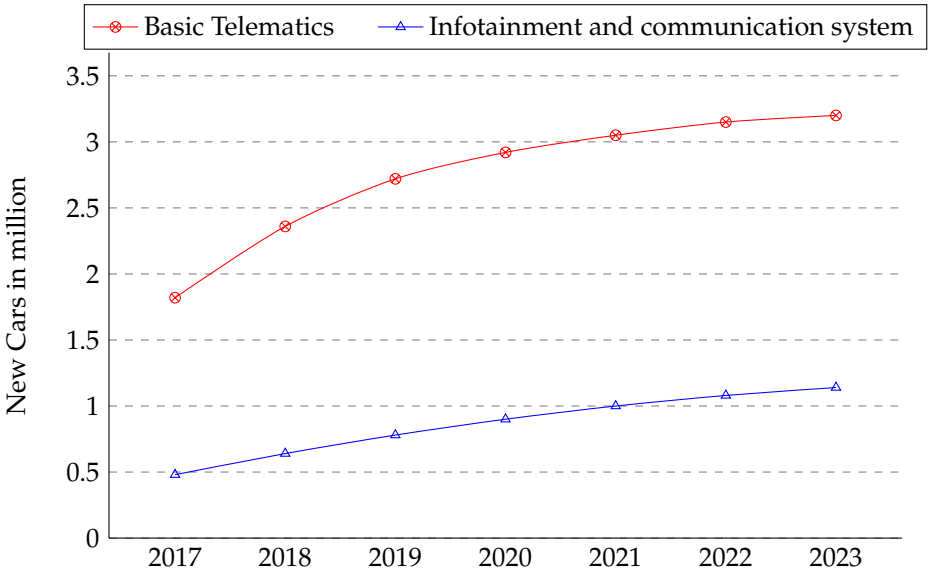


FIGURE 1.1: Estimated number of new cars (in millions) equipped with connected hardware in Germany from 2017 to 2023 by segment (based on [Sta18])

A modern car contains 100 - 70 micro controller-based computers (i.e., ECUs). Each ECU relies on a set of sensors and actuators to serve one or more of the E/E systems or subsystems in the vehicle, which range from the mundane such as controlling courtesy lights to highly critical applications such as engine control. These ECUs are grouped into various sub-networks based on their functions. The sub-networks are interconnected via a central gateway, and the ECUs within each sub-network communicate via different bus systems. Modern vehicles are also equipped with various technologies, such as WiFi, 5G, Global Positioning System (GPS), and Bluetooth, giving them the capability to collaborate and communicate with roadside units to ensure a safe and comfortable journey for drivers and passengers.

1.2 Challenges

Many issues and challenges make securing the in-vehicle system a difficult process. The aim of this section is to highlight some of these challenges, which we attempt to resolve in this thesis.

1.2.1 Safety vs Security

Premium vehicles include a huge number of software components which themselves include millions of Lines Of software Code (LOC). For example, the radio and navigation system alone in the 2009 S-class Mercedes-Benz has over 20 million LOC [Cha09]. Figure 1.2 shows that a modern vehicle includes more software code than a Boeing 787 Dreamliner, Mac OS X "Tiger", and more. The enormous number of software components within the vehicle are usually written by different teams with widely varying degrees of

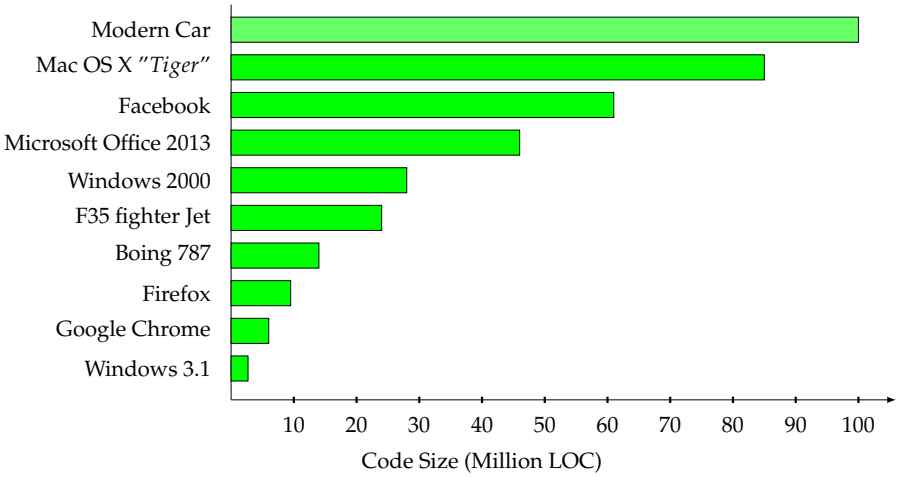


FIGURE 1.2: Size of software code used to develop a modern vehicle compared with code size of other different systems (based on [DES17])

competence. Integrating different components with high disparity in code quality into the same vehicular network can end up leaving the entire vehicle system in a non-secure state. The problem is that separating good quality code (safety-critical) from poor quality code (non-critical) does not in itself provide security since the two types coexist and share the same vehicular resources (e.g., network). Compromise of non-critical components by an adversary could be sufficient to enable him to control critical components across the entire car [Laa+09].

A recent report [Syn18] revealed that out of 593 surveyed professionals from global automotive manufacturers, suppliers and service providers, slightly over half of organizations do not perform any security testing to detect security vulnerabilities within the software components used in the vehicles they produce. The functionality and safety testing techniques which are used to test vehicle components are insufficient for catching security vulnerabilities [Bay+16]. Failure analysis and statistical models have proven quite useful for identifying components that are likely to cause problems. But they do not address interference by malicious adversaries who base their attacks on the exploitation of improbable scenarios (e.g., race conditions, malformed inputs, etc.). Thus, these models are not able to detect even those vulnerabilities which are most well-known, such as buffer overflows [And72], which have recently been discovered in vehicle components and exploited [Che+11].

All this shows the need to develop systematic mechanisms to ensure the security of the vehicle components, similar to those used to ensure safety. Developing such mechanisms requires determining the security requirements, vulnerabilities and threats which the system faces as well as the attackers who might target it. However, extracting such information is still not very detailed or standardized.

1.2.2 Heterogeneous Connectivity

Historically, the intra-vehicle network was designed without considering security requirements. Most of the communication is exchanged either in clear text or over a link with weak security, which means that malicious third parties can listen to or even inject false data or commands into the communication channel [Che+11]. The unrestricted interaction between components of mixed criticality, which can be hosted in different ECUs and allocated to different sub-networks, may also create vulnerabilities from a security perspective [Rou+10]. The attacker who can hack one ECU can apparently hack other ECUs, since they share the same network.

The assumption was that an attack on the in-vehicle network implied access to the vehicle, so physical security measures reduced the risk of such an attack. This assumption is becoming progressively disconnected from reality because of the different technologies now integrated within the vehicle to enable interaction with the external environment (e.g., GPS, digital radio, Bluetooth, etc.). These technologies have left internal networks which have weak or non-existent security mechanisms facing new threats of large-scale attacks. Attackers have been able to use existing security weaknesses in these technologies to intercept the connection between different ECUs, spoof the transferred data, and emit false data [Kos+10]. In some circumstances, attackers were also able to participate in communication by fraudulently using the identity of a legitimate ECU [Rou+10; Lab18]. All these attacks took place remotely without any need for physical access to the targeted vehicle.

As a result, securing in-vehicle communications has become a fundamental requirement. However, adding security to the existing communication protocols within the vehicle is not straightforward because of the resource constraints of ECUs, which might prevent the use of such mechanisms. Furthermore, the original design of some protocols does not consider security. Therefore, bringing in security for such protocols introduces performance overheads and increases the required packet size, which contrasts with the original goals behind designing the protocol in the first place (e.g., the Controller Area Network (CAN) protocol). Developing more effective and secure protocols is one important step towards eliminating a broad class of security attacks which threaten in-vehicle networks.

1.2.3 Security Policy Development and Enforcement

To keep security attacks from spreading across in-vehicle components, it is imperative to control the communications between components by defining a security policy which determines *who should talk to whom*. The huge number of ECUs as well as the software components which are mapped to them makes creating such a security policy a very difficult task, requiring detailed knowledge of the behavior of each component and all possible interactions between all potential system components. The process becomes even more challenging when considering an evolving and updatable system, in which component interactions may change, or new components may be added. Any errors during the configuration of this communication policy

may lead to mistakes that prevent the vehicle from functioning as intended, or worse, create some new security vulnerabilities that potential attackers can exploit.

Enforcing the security policy for this huge number of components is another challenge. Using the central gateway as an evaluation and enforcement point to authorize the various communications between ECUs in the different sub-networks is the most common proposal [WWP04; NXP18]. However, such a solution is not optimal, as the central gateway becomes a single point of failure. Any malicious software components in an ECU could be used to target the central gateway and drain its resources and, at worst, cause it to stop working, leaving the vehicle without any access control protection.

1.2.4 Continuous Monitoring

Cars are designed to operate for many years, during which time many new technologies could be introduced. With each new technology or application launched, new attack vectors are introduced. A static detection mechanism is thus inadequate since it will always be one step behind. Dependence on continuous updating of the security rules whenever new threats appear may not be practical, especially without the full adoption of a secure Over-The-Air (OTA) update framework. Also, using predefined rules to authorize or prevent a particular communication between two software components is insufficient as a security solution when it is used alone. Consider the case when one of these components is compromised; it would be still able to attack other parts as long as it attempts to compromise only those components with which it is authorized to communicate based on the security rules. From the point of view of policy enforcement, this component is not carrying out any malicious activities.

All this shows the need for behavioral-based monitoring mechanisms to detect and predict the off-nominal behavior of vehicle components. Defining the nominal behavior of the different components correctly is the main requirement for use of such mechanisms. Using inaccurate or the wrong behavior as a reference for off-nominal behavior detection may cause the monitoring system to consider a component that is operating nominally as malicious, or to identify a malicious component as benign.

1.2.5 Tighter Security vs Availability

Security can never be absolute. Even if all security mitigation is adopted, it is still possible for an attack to happen. The embedded nature as well as the safety-critical aspect of vehicles makes the response to a detected attack as critical as the detection of the attack itself. The typical system response when detecting an attack on a component is to restart that component [Str+09] in the hope that the failure (which was caused by the attack) was a transient one. Disabling a perfectly functional component because of a violation, such as an overflow, may have spectacularly unintended consequences; as the crash of the inaugural flight of the Ariane V booster demonstrates [LL96].

The heavy-handed security system may adversely impact availability. Adopting such security system leads to a high rate of false positives affecting availability and overall performance. However, once a software component is found to have violated its security boundaries, the system needs to take some remedial action. The security policy of the vehicle must implement different strategies in order to react to an attack. These strategies need to ensure high system resilience and safety. Generally - and even more so in the vehicular domain – intrusion response strategies have thus far received less attention and research efforts compared to intrusion-detection mechanisms [SBW07].

1.3 Contributions

All the aforementioned issues show the need for a holistic solution which ensures the security of the vehicular system during its whole life cycle, starting from the secure development of components, continuing to protect them while they operate, and reacting properly even when an attack is successful. The contribution of this thesis is to develop a comprehensive secure framework which aims to ensure the prevention and detection of, and response to, security intrusions which may face the vehicle system. The work of this thesis was achieved as part of the Controlling Concurrent Change (CCC) project [CCC13], which was funded by the German Research Foundation (Deutsche Forschungsgemeinschaft) for 6 years starting from April 1, 2013. The contribution of the thesis can be detailed as follows:

- Revising the existing vehicle-related threat modeling efforts to develop a comprehensive threat model which can be used to classify all conceivable attacks against the vehicular domain as well as an aid to illustrate attack vectors which threaten the different assets of the vehicle.
- Developing a framework to formulate the security policy for intrusion detection and response systems gradually by integrating it through the design and life cycle of software components to preserve the intentions of the initial designer and ensure the requirements of the actual operational platform.
- Developing a lightweight distributed access-control framework which allows only authorized components to interact with each other inside the vehicle and with external entities.
- Integrating and implementing a security protocol to ensure the integrity and confidentiality of in-vehicle communication as a part of the distributed access control framework.
- Developing an efficient host-based intrusion detection mechanism for in-vehicle systems. This mechanism is used to identify the proper temporal thresholds for every safety-critical software component to give the system the ability to predict the presence of malicious activities.

- Defining effective strategies to respond to security violations with consideration of issues such as containment, continued availability, interaction with other subsystems, and, in certain cases, latency.

1.4 Organization

This thesis is divided into three parts. The first part provides a landscape for the thesis via three chapters; chapter 2 gives background information about in-vehicle systems and the different vulnerabilities and attacks which affect them, as well as the defense mechanisms which were proposed to mitigate these attacks. Chapter 3 presents the software and hardware setup and the use cases which are employed throughout the thesis. In addition, it introduces the proposed multilayer framework for securing the vehicle.

This framework is detailed in the second part of the thesis, which includes four chapters. Chapter 4 details the proposed threat model. Chapter 5 explains the mechanism for constructing the security policy for the in-vehicle system and how this policy can be enforced. In chapter 6, we demonstrate how the temporal properties of some applications can be used to derive an intrusion-detection schema. Chapter 7 outlines how we can develop an intrusion response for the in-vehicle system through the collaboration of all previous proposed mechanisms in chapters 4, 5 and 6.

Chapter 8, which is the only chapter in the last part of the thesis, summarizes the work in this thesis. It also contains an outlook on future work on the topics covered in the thesis. Finally, a list of publications used during this thesis, Figures, and Bibliography can be found.

"Nothing is particularly hard if you divide it into small jobs."

Henry Ford

2 Foundations

This chapter aims to give an overview of the basic concepts required to understand the following chapters. It also presents some of the most relevant and influential research that we use as a foundation for the work in this thesis. Section 2.1 describes the primary components which compose the in-vehicle system. In Section 2.2, we discuss the vulnerabilities and the attacks which could target each of these components. Finally, Section 2.3 details the mitigation mechanisms which were adopted to protect the in-vehicle system and address the explained attacks.

2.1 Vehicular System

Recently, vehicle manufacturing has changed significantly. These changes are reflected in the increased use of automotive embedded systems and embedded software components, which are integrated into every single vehicle. A modern vehicle may contain up to 100 microcontroller-based computers, known as ECUs, which run huge codes [Bro+07; Cha09]. Each ECU relies on a set of sensors and actuators to serve one or more of the E/E systems or subsystems in a vehicle. Different types of communication buses are used to interconnect the ECUs distributed inside the car. In this section, we will discuss the ECU and what kind of software it runs, as well as how it is networked within the vehicle.

2.1.1 ECU

The introduction of ECUs was the revolution which moved the vehicle from a mechanical to E/E system. Each modern car contains a massive number of ECUs. For example, a car like the Volvo S80 comprises more than 30 embedded processors. Similarly, BMW 7- Series and the Mercedes S-class both include over 60 ECUs [Tur02]. Each of these ECUs is an embedded

computer which is used to control the mechanical and electrical components within the car. Based on the elements they manage and the software they run, these ECUs can be categorized, from safety-critical ones such as the emission-related control ECU to non-critical ones such as the infotainment ECU [Bos86].

The hardware properties of vehicles' ECUs also vary from one ECU to another. Generally, ECUs come with limited resources concerning computational power and memory size. Most of these ECUs are able to handle a single vehicle functionality (one function per ECU). The trend nowadays is to merge multiple ECUs into a few ECUs with more powerful multi-core processors, ample memory, and the capability to serve many functionalities together [DNSV10]. Such ECUs were already integrated within modern vehicles for infotainment systems. The goal behind using such powerful ECUs is to reduce their overall number, which is growing due to the increase in newly introduced functionalities. It also aims to limit the complexity of intra-vehicle networks. Moreover, using fewer ECUs helps to reduce vehicle weight, power consumption, fuel consumption and production cost. However, introducing such ECUs has its own drawbacks which need to be considered [EK13].

2.1.2 Software Components

Today, a huge number of software components are running over the various ECUs aiming to improve the safety, efficiency and comfort of modern vehicles. In premium cars, we can find up to 2000 software-based functions [Bro+07]. This number keeps increasing day by day. Thus, the size of software code is also growing. For example, the radio and navigation system in the 2009 S-class Mercedes-Benz alone has over 20 million LOCs [Cha09].

These software components include regular automotive applications as well as the firmware which support these applications.

Automotive Applications

Modern vehicles are full of software applications. These applications differ from each other in terms of the functions that they perform and the nonfunctional requirements (i.e. security, reliability, performance, and so forth) that they need [HZ+18; Bro+07]. In general, those applications can be grouped into two main categories:

- **Safety-critical applications:** which are used to control critical functions. The failure of these applications could affect the safety of the vehicle's occupants and may led to catastrophic consequences. Examples of such applications are powertrain, chassis control, head-on collision warning system, Adaptive Cruise Control (ACC) system, and control loss warning system. Those applications have certain requirements in terms of latency (called hard real-time applications, as we will see in Chapter 6), error probability, and strict safety and availability requirements.

- **Non-safety-critical applications:** which include all other vehicle applications. Mainly, applications within this category aim to improve driver comfort and enhance traffic efficiency, such as multimedia and telematics applications. These applications usually have more relaxed requirements compared to safety-critical applications.

It is crucial to note that the safety-based categorization of applications has no impact on their required security level. Software components need to be developed with intensive security considerations, whether these applications are safety-critical or not. This categorization can provide the system designer with valuable information about where she can introduce another level of security to protect highly critical applications.

Firmware

Firmware is the special class of software components which are used to control the various hardware devices of the embedded systems (i.e., ECU) and to provide a foundation layer to run automotive applications. Firmware for embedded devices is like the Operating System (OS) in standard computers. However, firmware often contains a special purpose OS. This OS is often a real-time microkernel OS designed to serve the different automotive applications and support them to meet their time constraints.

The microkernel OS is one OS type which has some unique properties. In contrast to a monolithic OS, in which all OS services share the same address space and run in kernel mode, the kernel in microkernel OS includes the minimum amount of services and features required to implement an OS. Other services, such as device drivers, file systems, and networking stacks are run alongside the applications in user mode. Each one runs in a separate address spaces isolated from the kernel and other services and applications.

OSEK/VDX ("Offene Systeme und deren Schnittstellen für die Elektronik in Kraftfahrzeugen/Vehicle Distributed Executive" in German, translated to English as "Open Systems and their Interfaces for the Electronics in Motor Vehicles/vehicle distributed executive") standard [Isof] defined a microkernel *Real-Time* Operating System (RTOS) (known as OSEK OS) which has been developed in many ECUs to provides an architecture for distributed real-time applications in vehicles.

AUTomotive Open Systems ARchitecture (AUTOSAR) [AUT16] is a widely used, open and standardized software architecture for automotive domain. It aims to support the independent development of the automotive application by abstracting the underlying ECU hardware. AUTOSAR deploys three layers comprising the application layer where the automotive applications can run, and the Run-Time Environment (RTE) layer which provides communication services for all applications to exchange data in the same ECU or with other applications mapped to another ECUs. The final layer, called the Basis Software (BSW) layer, contains the AUTOSAR OS. AUTOSAR defined an operating system called AUTOSAR OS based on OSEK OS [AUT14].

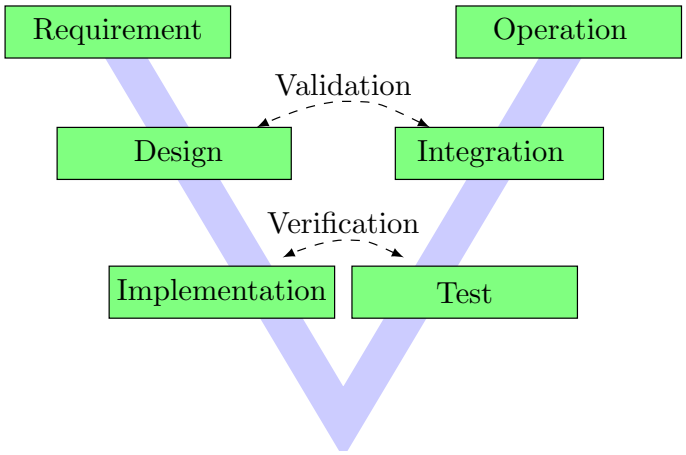


FIGURE 2.1: Simplified V-Model with the different phases of product development

Special purpose OSs, such as AUTOSAR OS and OSEK/VDX, are not the only OSs which run over the various ECUs of the modern vehicle. Other general purpose OSs such as Windows Embedded Automotive, Android, and Linux-based OSs are also adapted to run non-critical safety systems [CM16]. For example, the entertainment system in Tesla vehicles runs an embedded version of Linux [Yon14]. Other vehicle manufactures (specifically Volvo, Nissan and Mitsubishi) have adopted a special version of Android OS called Android Automotive to be used for their modern vehicle entertainment systems.

Development Process

The increase in the number of software components within a vehicle has made the product development system of new vehicles a very complex operation, especially if we consider how many partners participate distributively in the process. Thus, the adoption of a development process becomes a necessary and important factor in the success or failure of a new automotive product [HR10]. Generally, the standard V-model software development process is adopted in the automotive domain [Isoi]. The "V" name comes from the basic shape of the model as shown in Figure 2.1. This model extends the waterfall model [FM91] by determining the relationships between each phase of the development and its associated phase of testing.

The V model follows the top-down development approach. Development starts at the upper left side of the model with the requirements analysis phase, when all the functional and non-functional requirements of the system are collected and defined. This phase is followed by the design phase, in which the system is designed based on these defined requirements. The last phase in the left branch is the implementation phase, during which the actual coding of the designed system takes place. The right branch includes progressive verification and validation operations which start by testing each software component, then go on to test the entire system after integration of

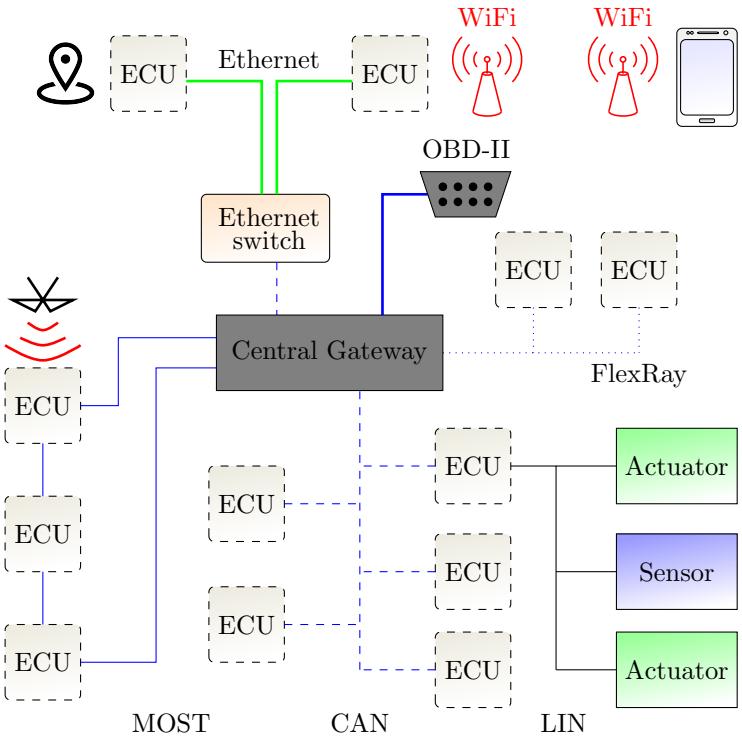


FIGURE 2.2: Vehicle network architecture influenced by [Nol06]

the different subsystems to ensure the commitment of the defined requirements, before releasing the vehicle.

2.1.3 Network

The different software components which are hosted by those ECUs are required to collaborate with each other. Therefore, many ECUs are collected together in various sub-networks and share the same bus system. Within each sub-network, a certain type of bus system and communication protocol is used to enable communication between the different ECUs. To access the status of the different sub-networks from outside, a mandatory port called On Board Diagnostics (OBD-II) port is integrated into all new vehicles. OBD-II is located under the dashboard of the vehicle and used for maintenance and diagnosis purposes.

Figure 2.2 shows one of the current in-vehicle network architectures where a central gateway is used to enable the exchange of data between the different incompatible sub-networks. Nowadays, another network architecture is becoming more popular. This architecture is based on dividing the automotive network system into many independent sub-networks; each group's components (ECUs, sensors and actuators) belong to the same functional domain, such as the entertainment domain, body domain, or powertrain domain [Lim+11; HSM12]. A local gateway is used within each domain network. Those gateways are interconnected with the central gateway via Ethernet links as shown in Figure 2.3.

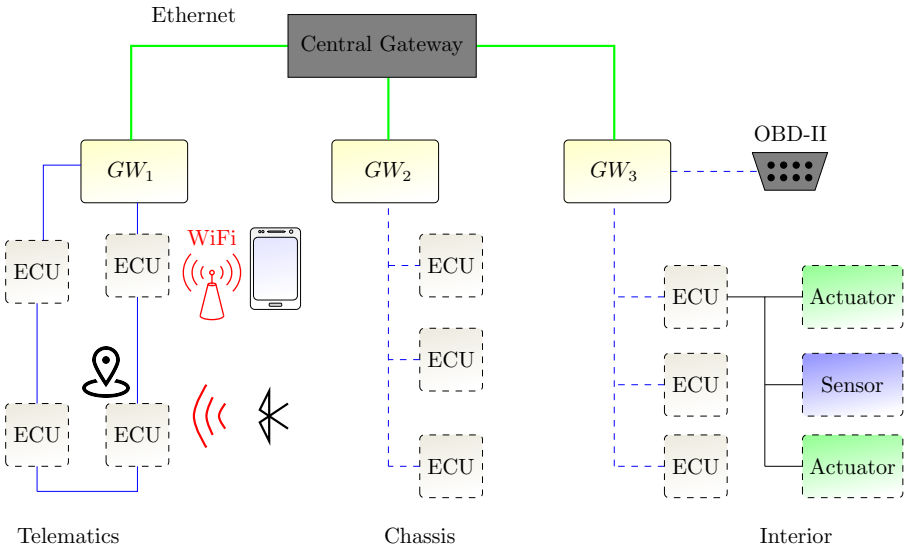


FIGURE 2.3: Domain-based vehicle network architecture, based on [HSM12].

However, there is no fixed architecture for vehicle networks. It varies between the different manufacturers and among models of the same manufacturers. In the following, we introduce the most common bus systems which are used within the different vehicle sub-networks. More technical information about these systems can be found at [WWP06; Tuo+15; ZKC16; NSL13; NSL08].

- **CAN:** the most common bus system, which is used to interconnect ECUs, actuators and to receive feedback from sensors via a single or dual-wire [Isoh]. It was introduced in the early 1980s. CAN is a low cost and reliable bus system, but it provides low bandwidth (up to 1Mbit/s). CAN is used in powertrain, chassis, and body electronics. CAN is used to connect applications that need real-time communications.

- **Local Interconnect Network (LIN):** developed in the late 1990s, LIN provides low-cost, serial communication between automotive smart sensors, and actuators with data rates up to 20 Kbit/s [Isoe]. LIN is likely used for body electronics such as mirrors, seats motors, accessories, door locks and climate sensors.

- **FlexRay:** introduced by the FlexRay consortium, a group of car manufacturers and Original Equipment Manufacturers (OEMs), including Robert Bosch GmbH. It was introduced in 2004 and, later adopted as ISO standard 17458 [Isog]. FlexRay ensures faster communication compared to CAN and LIN (bandwidth is up to 10 Mbit/s). It is used to connect safety critical applications which require predictability and fault-tolerance. FlexRay has two parallel channels; one is used as a backup in case of any communication failure in the other. FlexRay is used in high-performance powertrain and safety features such as drive-by-wire, active suspension, adaptive cruise control, etc.

- **Media Oriented Systems Transport (MOST):** developed in 1998, the last revision of this protocol, called MOST150, was introduced in 2007. MOST is a high-speed bus system which was developed for vehicular multimedia and infotainment applications which require transmission of a large amount of data. The bandwidth of MOST150 is 150 Mbit/s. It is only used for camera or video connections.

- **Wireless-based:** Wireless communication is used to connect different sensors and actuators within the in-vehicle networks. For example, each high-tech vehicle has a safety system called the Tire Pressure Monitoring System (TPMS) which is used to monitor and display air pressure inside the vehicle tires. TPMS depends on short-range wireless communication to receive the tire pressure reading from the tire sensor. Bluetooth is another wireless technology which is already used within the vehicle to support some luxury applications such as hands-free phone calls. WiFi is another technology which is used to connect user devices such as phones and tablets to the vehicle.

  Long-range wireless technologies are also used in vehicles. The most common applications which deploy these technologies are the GPS receiver which is used to receive positioning information, the Traffic Message Channel (TMC) receiver, which decodes the TMC data about the traffic information and displays it in the navigation system map, and digital radio.

- **IP and Ethernet-based:** Day by day, vehicular applications are becoming more complicated and interconnected. Consequently, the need for these applications to exchange bigger and more expressive messages is increasing and pushing towards using IP and Ethernet-based standards for both on-board communications and Vehicle-to-everything (V2X) communications [Ste+12; Tuo+15; Cum+12]. Ethernet is able to transverse data two orders of magnitude faster than CAN. In addition, Ethernet provides higher bandwidth data transmissions compared to other bus systems.

  Many authors expect that Ethernet will replace other in-vehicle bus systems in the near future [HSM12; HZ+18] similar to the case in avionics where a modified Ethernet adoption is already happening [Ale+07]. To achieve that, there is huge ongoing work to fill the gaps involved in full adoption of Ethernet, such as supporting real-time safety applications. [Ste08; SZ18]. Until now, automotive Ethernet has been used for interfacing with diagnostic equipment [Isod], data-intensive applications such as the entertainment systems [Sau14], and camera-based driver assistance systems.

In this thesis, we consider the use of IP and Ethernet-based communication links between the different ECUs.

2.2 Automotive Security

2.2.1 Security Attributes

The CIA (**C**onfidentiality, **I**ntegrity, and **A**vailability) triad represents the three most essential security attributes for any IT system.

- **Confidentiality:** having this property in a system means that only authorized parties of that system can manage to disclose the system information [Shi07a].
- **Integrity:** refers to the ability to ensure the consistency and trustworthiness of system information [Shi07a].
- **Availability:** means that the system information and resources remain accessible and usable upon demand by authorized parties [Shi07a].

2.2.2 Vulnerabilities and Attacks

The literature includes several studies (e.g, [MV14], [HZ+18], [SDK19], [Stu+13], and many others) which investigate automotive security and highlight the various vulnerabilities within vehicle systems. In this section, we show security vulnerabilities and attacks within each of the main parts of the automotive system (i.e., ECU, software components and network). The danger of exploiting any of these vulnerabilities is not limited to the components in which the vulnerability exists, but extends to cover other components and could put the entire system at huge risk. A simple buffer overflow vulnerability in one software component run in a vehicle can be an entry point for attackers to control the entire vehicle.

ECUs and connected devices: ECUs are programmable devices which include some ports and serial consoles to help the developer access and maintain the firmware and software mapped to this ECU. The same ports and consoles can give attackers the ability to re-flash the ECU with malicious custom firmware [BAG15]. In 2015, Charlie Miller and Chris Valasek [MV15] were able to flash one of Cherokee Jeep's head unit chips with modified firmware (*Chip tuning attack* [WW15]). Later, they used the malicious firmware to send commands through the in-vehicle network to perform malicious actions such as disabling brakes, taking control of the steering wheel, and even stopping the engine. Another primary attack against the ECU is a *side channel attack*. By carrying out such attacks, the attacker can gather information during the execution of the build in crypto-system and use the information gathered to extract secure critical information such as crypto keys [Koc96; SK14; Eis+08].

ECUs are not the only hardware components which could be targeted by attackers. Other devices which are connected to the vehicle, such as phones, tablets, diagnostic devices connected via ODB-II, etc., can be used as a gateway to attack a vehicle if it contains a security vulnerability. Woo et al. [WJL15] demonstrated the possibility of attacking a vehicle remotely

by using a malicious application installed on a smart phone connected to the victim's vehicle.

Automotive Software: The massive size of the critical and noncritical software components within each vehicle make the existence of security vulnerabilities is not unusual. Although both critical and noncritical safety applications must consider security during development, this is not always the case in reality. Vehicular software components are usually written by different teams with widely varying degrees of competence; some automotive organizations do not even have any product cybersecurity management programs [Syn18].

Integrating different components with highly disparate code quality into the same ECU could end up leaving the entire vehicle system in a non-secure state. The compromise of non-critical components by an adversary could be sufficient to control critical components across the entire car [Laa+09].

Besides, using unsafe programming languages (such as C which is considered defect prone [Ray+14; Ber+19]) has lead to the introduction of many vulnerabilities such as buffer overflow, dangling pointers and so forth. Such vulnerabilities have been the starting point for many successful attacks [Che+11; MV15].

Almost all vehicle vendors have suffered from security weaknesses within one or more of their software components. For example, there was a vulnerability in Chrysler's Uconnect software [MV15], Skoda's SmartGate system [Lin15], BMW's ConnectedDrive [FAS15], the WIFI access point of the Mitsubishi Outlander plugin hybrid electric vehicle (PHEV) [Lod16], GM's Onstar [Tho15], and many others. Those vulnerabilities gave attackers the chance to perform numerous attacks and many malicious actions such as turning on/off air conditioning, heating, and lights, disabling the theft alarm and so forth. They also caused the recall of millions of cars.

Not only automotive applications but firmware itself can contain many vulnerabilities or unnecessary services which could be exploited by an attacker [Yon14; Dun13]. For example, Tesla S was running an Ubuntu-based OS with opened ports for unnecessary functions such as telnet, SSH and so forth which were employed by users to connect to the vehicle's system [Yon14].

OTA firmware and software updates represent a new challenge as well as a predictable resource for introducing new vulnerabilities to the automotive system if not handled correctly. One malicious or wrong update can end up as a huge issue, as in the case of the 2016 Toyota Land Cruiser Enform system, which one ECU was continuously rebooting itself because of a new update containing errant data [Bog16].

In-Vehicle Network The notion of securing the internal vehicle network was based on the fact that the vehicle is physically protected. The only way to hack a car was by being inside it and using one of the external interfaces, which give the attacker direct access to the internal network. ODB-II ports

used to be the most famous interfaces which gave the attacker such direct access. Other interfaces such as CD/DVD players or USB ports (which are used to connect drivers' smartphones) also used for the same target [Che+11]. However, the newly introduced technologies which enable the car to interconnect with the outside world have made the physical isolation of the internal vehicle network history. Nowadays, attackers are able to exploit various vulnerabilities with the automotive system, and hack and control the vehicle remotely.

Traditionally, internal vehicle bus systems have been developed to fulfill safety, cost and efficiency requirements without considering any security risks. This is clear if we consider how the CAN bus system operates. CAN messages are broadcast and transmitted in clear text without any real evidence of data integrity or authenticity [Kos+10]. Consequently, any malicious ECU can pretend to be any other legitimate ECU (i.e., *masquerade attack*), intercept all the messages exchanged between the different ECUs (i.e., *man-in-the-middle attack*), manipulate the transmitted data (i.e., *spoofing attack*) or fraudulently delay or re-transmit previous messages (i.e., *replay attack*). In addition, the malicious component can flood the bus with fake high priority messages to disrupt other communications (i.e., *Denial of Service (DoS) attack*) [MV15].

However, these issues are not limited to the CAN bus; other bus systems (e.g., Ethernet [NLC14], FlexRay [Nil+09], LIN [Tak+17]) suffer from similar deficiencies which have also made them targets of many attacks.

The wireless network, which is used by multiple applications in the vehicle, is not any better when it comes to security. Many attackers have targeted systems which depend on wireless communications. For example, Roulf et al. [Rou+10] investigated the TPMS. They found that the sensors transmitted messages without any security protection. Therefore, attackers on the roadside (up to 40 meters away) or driving another vehicle can eavesdrop, intercept and inject spoofed messages by using a low-cost device. Other evidence showed that the attacker could open vehicles without the key [FDC11; Tho15] by replaying wirelessly transmitted messages between the car and the smart key.

2.3 Security Mitigation

During the last few years, providing security for the automotive domain has been a hot topic and received considerable attention; this is expected to remain the case. Many national and European projects, such as CCC project [CCC13], SeVeCom [Lei+06], EVITA [HRS09], PRESERVE [Con+14], SHARCS [SHA15] and many more were funded to investigate this topic and invent solutions for the various aforementioned and future issues related to vehicular systems.

In this section, we surveyed these efforts and classified the proposed mitigation methods into three lines of security defense: prevention, detection and response. This classification is influenced by other research such as [NL09; HZ+18].

2.3.1 Prevention Mechanisms

Prevention mechanisms represent the first line of security defense for the automotive system. The aim of these mechanisms is to prevent the occurrence of attacks proactively. These mechanisms include:

- Secure development of automotive software components to prevent the attacker from benefiting from software vulnerabilities.
- Developing hardware security mechanisms to support the reliable execution and isolation of these software components to prevent the attacker from controlling the entire system if she succeeds in compromising one part.
- Defining a secure link and a proper access control mechanism to support the secure communication of these components while they are running and prevent any malicious third party from interfering with their interactions.
- Ensuring a reliable framework to update these components to prevent malicious parties from introducing new vulnerabilities.

Next, each of these points is discussed in more detail.

Secure Software

Practically speaking, automotive software components form the main part of the automotive system and used to be a very attractive point for attackers. Many mitigation mechanisms have been used to improve automotive software security. These techniques are used throughout the different life-cycle phases of the software component, starting from implementation, and ending with the phase where the components are running on the desired ECUs.

The first step to prevent cyberattacks is by writing a secure code in the first place during the *implementation* phase. Writing vulnerability-free source code is not a straightforward process, and is often impossible, especially in the case of a large-scale system as in the modern vehicle. Writing secure code requires educating software developers to understand security practices in software development and teaching them how to eliminate common mistakes. However, even by doing this, there is no guarantee that even such well-educated developers will ultimately develop secure software [Nai+17]. On the other hand, since most software bugs result from human errors, secure code generation tools could be used to reduce human error interference. However, such tools are often considered inefficient for the automotive domain [Bro06b].

Testing the software component is a primary part of the development process (see Figure 2.1). Regarding ISO 26262 [Isoi], this test includes functional testing (i.e., verify that software does not perform undesired functionality) and safety testing (i.e., verify software fulfills the software safety requirements). The need to extend the testing methods by including a software security test became a crucial demand due to the many cyberattacks which

targeted the modern vehicle [Mar13]. Software security tests help to identify and mitigate security weaknesses before an attacker can exploit them in the field, causing catastrophic results. Many authors (e.g., [Bur+12; Bay+15]) have proposed the integration of security tests within the development process. However, using such security tests is still not very common within the automotive domain [Bay+16; Syn18].

Automotive software security tests can benefit from existing static analysis tools [CM04; Joh+13] and use them to scan automotive software, searching for hidden vulnerabilities. Such tools do not require the running of the software code, and thus can be used before the *integration* of the code into the system. A penetration test [ASM05] is another security test which can be performed to discover other security vulnerabilities and determine the resistance level of the system.

One important mechanism to prevent the attack from spreading from one software component to another while they are running on the same ECU is by isolating the different components from each other [Stu+09; Gu+12]. Virtualization aims to divide the ECU resources into isolated RTEs. System virtualization could be achieved by means of different technologies such as the hypervisor [PWW08] or containerization [Mor+17].

Hypervisor-based virtualization depends on the emulation of the platform resources by a software called hypervisor, and sharing these virtual resources many times with multiple instances of Virtual Machines (VMs) which run simultaneously. There are two types of hypervisor-based virtualization: *bare metal* where the hypervisor runs directly on the hardware, and *hosted or OS level* where the hypervisor runs as a typical application on the top of a host OS [AH10]. Container-based virtualization does not require the emulation of the platform hardware. It depends on the OS creating an isolated environment for different applications which run on top of it. All these isolated applications share the same resources of the host system. Many research papers provide a performance comparison between the traditional hypervisor-based and container-based virtualization approaches as well as the drawbacks and cons of using each of them [MKK15; Hei08; Mas+16; Ede16].

Although the aforementioned isolation mechanisms ensure sufficient protection for the application from other distrusted applications, they do not protect its critical data from malicious highly privileged components such as the host OS or the hypervisor [JKB15]. Such issues pushed toward using hardware-based security solutions.

Hardware Security

The main goal of using hardware security mechanisms is to create safe and secure storage for critical data such as crypto secret keys. These mechanisms are designed with built-in defenses against physical side-channel attacks. EVITA project has developed such a solution for automotive domain. It is called the Hardware Security Module (HSM), which is used to store and process security-critical applications (mainly crypto operations) shielded from any other potentially malicious applications on the same ECU [WG11].

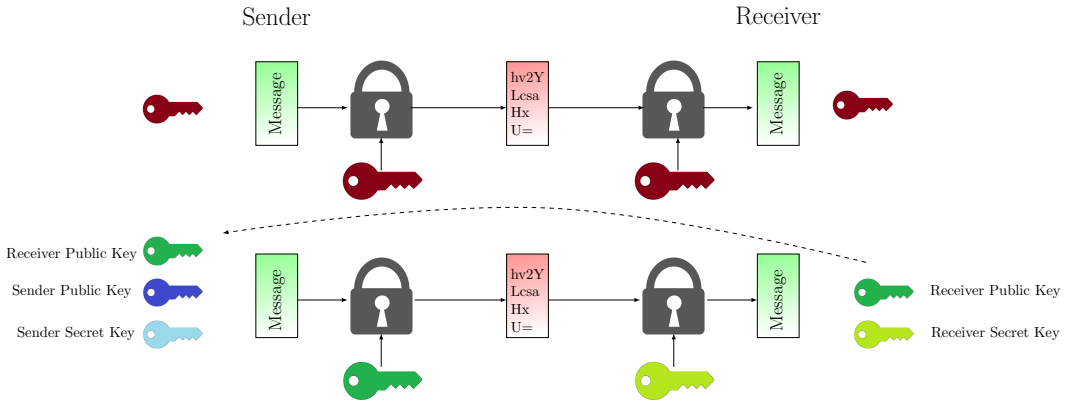


FIGURE 2.4: Symmetric (top) and asymmetric (bottom) encryption.

Another aim of using hardware-based security solutions is to detect and mitigate boot code infection. In such circumstances, the HSM can be used as a root of trust to authenticate the boot code before it runs. This include firmware as well as other critical applications [Idr+11]. Other solutions such as Intel Software Guard Extensions (SGX) [CD16] are introduced to guarantee the integrity of an application's code which runs on a platform where all the privileged software is potentially malicious.

In addition, hardware solutions are used to accelerate the execution of crypto algorithms which are used to support secure communications, as we will see in the next subsection.

Cryptography And Secure Link

Basics: One of the main approaches to boosting the security of cyberspace is by adopting cryptography, for example, to safeguard the confidentiality of data exchanged between the sender and a receiver. The message sender can use an encryption algorithm and a crypto key to translate the plain-text of the message into cipher-text, which she can send to the remote party. During its journey to its final destination, no third party can read or extract the information from the cipher-text, except if the third party knows the key and algorithm used. The message receiver uses the same algorithm and key to decrypt the cipher-text to obtain the same plain text which was created by the sender.

The encryption algorithms can be either symmetric (e.g., Data Encryption Standard (DES) [PUB77] and Advanced Encryption Standard (AES) [DR01]) or asymmetric (e.g., Rivest–Shamir–Adleman (RSA) [RSA78]) algorithms (see Figure 2.4). In symmetric encryption algorithms (a.k.a. secret key or shared key algorithms), both sender and receiver use the same pre-shared secret key to encrypt and decrypt the message, while in asymmetric algorithms (a.k.a. public key algorithms) both the sender and receiver have a distinct pair of dependent keys. One key, which is called the private key, is kept secret while the other, the public key, is made available to anyone. A

message which was encrypted by one of these keys can only be decrypted by the other.

Adopting Cryptography: Most of a vehicle's internal communications are exchanged in clear text, and applying encryption is still rare. This is because of many widespread beliefs; firstly, these algorithms, especially asymmetric ones, require high computational power which is not always available. Moreover, they introduce an overhead to the system either by increasing the size of the transmitted data or by increasing the system latency. However, in this thesis we will show how these beliefs are not totally correct.

Some authors have proposed frameworks which use symmetric encryption algorithms to protect the confidentiality of the in-vehicle communications over CAN bus [WJL15; Sid+17; GR09]. In other cases, e.g., [Mun+15], an asymmetric algorithm is used to encrypt the shared key which will be used by the symmetric algorithm to encrypt the exchanged messages later.

Another goal of using cryptography is to guarantee that the transmitted data is not changed (deliberately or unintentionally) in transit, and that it is transferred by the sender who it should be sent by. Message Authentication Code (MAC) [Can+99] can be used to ensure the integrity and the authenticity of the message. By using a MAC algorithm and a pre-shared key, as with symmetric encryption algorithms, the message sender can create a tag (called MAC) for each transmitted message and send it together with the message. The receiver uses the received message, pre-shared key, and the same MAC algorithm to compute another tag and compare it with the one she received from the sender. If both tags match, she knows that the message has not been altered and it originated from the specific sender who knows the key. MAC algorithms can be based on a symmetric key block cipher algorithm (Cipher-based MAC [Dwo05]), such as CBC-MAC [BKR00], or based on one-way hash function (Hash function-based MAC [BCK96; KCB97]).

Unlike encryption, MAC is adopted frequently to ensure the integrity and authenticity of in-vehicle communications [ZM14; Mun+15; LS+11; Sch12]. In [Lin+13; WJL15; Ham+08], authors have used CMAC algorithms to protect against masquerade and replay attacks on CAN bus, while [Woo+16; Ued+15] used HMAC algorithms for the same purpose. MAC was not used to protect CAN only; many authors have proposed the use of HMAC to ensure the security of the communication over FlexRay bus [Han+14; Mou+16].

One of the main issues that MAC suffers from is the non-repudiation problem. Digital signature sachems solve this issue by using public-key cryptography. The sender of messages can cryptographically sign the transmitted data using her own private key (in practice, the sender signs the computed hash of the message only) so the receiver can guarantee the message's integrity and authenticity by checking the validity of the signature using the sender's public key.

Many authors, such as [WWP04; Idr+11], have proposed frameworks which use a digital signature to provide message integrity and sender authentication for the intra-vehicle network. However, using a digital signature is not applicable for protecting CAN messages because the signature

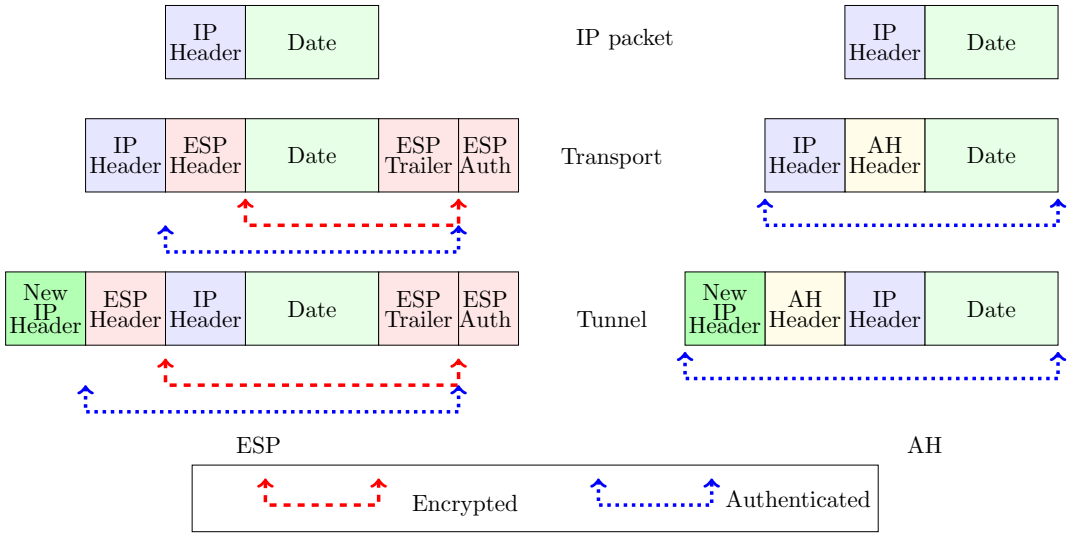


FIGURE 2.5: IP packet with IPsec fields in both modes and protocols.

produced is too long to fit in one message, therefore, securing one message requires several messages [HL18; FL15a].

AUTOSAR has introduced many APIs for crypto services which can be used by the software components [AUT17]. These services include specifications for calculating of hash, creating and verifying MAC value and digital signature, and achieving symmetric and asymmetric encryption and decryption operations.

Internet Protocol Security (IPsec): As we have mentioned before, IP/Ethernet represents the future for in-vehicle communication. Besides the huge bandwidth and large payload size that it supports, IP/Ethernet brings real opportunities to improve the security of in-vehicle (as well as Vehicle-to-Vehicle (V2V)) communications by benefiting from the significant security expertise that was gained during the many years of using IP/Ethernet for traditional IT systems.

IPsec protocol [SK05] is one of the standards for protecting IP traffic. It was designed to ensure the integrity and confidentiality of each IP packet transmitted in the communication session. IPsec runs in the IP layer, which makes it transparent to upper layers, and therefore applications do not need to have any knowledge of IPsec to be able to use it.

IPsec is based on two encapsulation protocols. These are Authentication Header (AH) [Ken05a], which provides data-origin authentication, data-integrity, and protection from replay-attack, and Encapsulating Security Payload (ESP) [Ken05b] , which provides data confidentiality and also supports its own authentication scheme like the one used in AH. IPsec can be used in either transport or tunnel mode [SK05]. Transport mode is used to protect the data flows between a pair of hosts, while tunnel mode is used to protect the data flows between a pair of hosts, two gateways, or between a host and

a gateway. In addition, the implementation of each mode is different. Figure 2.5 illustrates both AH and ESP protocols in transport and tunnel modes.

Since the adoption of IP/Ethernet-based communication is still limited for in-vehicle communications, little effort has been made to investigate the use of IPsec to secure such communication except for a few exceptions such as [HP15; LH15].

In Section 5 of this thesis, we extend the implementation of an embedded version of IPsec and adapted it to ensure secure communication between the different ECUs within the vehicle.

Firewall

One of the most famous mechanisms for preventing network attacks is a firewall. Within the traditional network systems, a firewall is used to monitor and filter inbound and outbound network traffic based on predefined security rules, preventing unauthorized nodes from receiving and sending data [KP07; CBR03]. The need for such a system to control the interaction between the components within the vehicle and between the vehicle and the outside world has been addressed by many authors [Kos+10; KOJ11; Stu+13].

The location in which the firewall is placed and the communication that it should control were the main differences between the various proposals. Since the firewall is used as a boundary between two networks, the automotive firewall was introduced in many proposals, such as [Zre+08], to control the communication between the vehicle and outside world. Although protecting the vehicle from external threats is important, leaving the internal network of the vehicle without control has serious consequences.

Chutorash [Chu00] went a step further when he proposed using a firewall to control certain interactions within the vehicle. His approach was restricted to monitoring the communication between human-machine interface systems and other vehicle components only, while it ignored the other interactions between the different vehicle components.

Another solution was based on using the central gateway as a firewall to intercept the communication exchanged among the different bus systems [WWP04; NXP18]. All the security keys to encrypt the links between the different sub-networks as well as the access control rules for these connections are stored within this gateway. However, the central gateway provides an attack surface which may be considered a single point of failure. Moreover, the communication within each sub-network is not controlled. To cover these limitations, another solution was introduced to use the firewall within each sub-network gateway (these gateways are connected to the central gateway) [Ala18].

In this thesis we extended this approach by deploying a firewall for each single ECU in order to build a distributed firewall which is concerned about all communications inside the vehicle as we will see in Chapter 5.

Security Policy

The set of predefined security constraints which determines which actions are authorized and which are not and need to be adhered to by the system, is called the security policy [GR95]. One example of security policy is the Access Control List (ACL) [Shi07b] which is enforced by the firewall. Each rule in the ACL determines which network traffic should be allowed and which should be denied based on certain characteristics that can be extracted from each network frame, such as the address of the IP/Ethernet frame. ACL was part of many security frameworks proposed for securing in-vehicle communications, such as in [GR09; PDB14; Rum+19].

Developing, generating, and maintaining such an access control policy within the automotive domain is a labor intensive and error prone process. How to express the security policy is a very crucial point. Many security policy languages were proposed to enable the specification of the security policies for automotive systems. XACML language [Xac], PNL [IR12] and Abbreviated Language for Authorization (ALFA) [GNB15] are examples of these languages which were used to describe the access control rules in the different frameworks. To benefit from the existing modeling languages, such as UML, some authors (e.g., [Ber+17]) have proposed extensions to such languages to enable the integration of security specifications for the communication between the software components.

Blaze et al. [BFL96] invented the concept of trust management to solve access control problems in distributed systems. Ioannidis [Ioa05] has adopted has provided a framework which ensures the consistencies of the huge number of distributed security polices as well as the distributed enforcement of these polices. They proposed a language (i.e., KeyNote language [Bla+99]) to express the access control rule of the security policy. At the same time, KeyNote provides several advantages, such as the delegation mechanism. Many authors adopt the trust management concept to implement an access control framework in diverse fields, such as web applications [Chu+97], distributed firewall [Ioa+00], and others.

In this thesis, we used KeyNote trust management system to support the development of in-vehicle components security policies (see Chapter 5)

Secure Over-The-Air Update

OTA Update was introduced in the automotive domain to eliminate the need for the vehicle's owner to visit the dealership to install a new patch or software. The dark side of this process is that it increases the attack surfaces of the vehicle and creates a new attack vector for hackers. To prevent that, a security mechanism should be adopted to make OTA update more reliable and secure.

Digital signature schema plays a major role in securing the OTA software and firmware update process [Kar+16; MTK19; NL08]. With the help of digital signatures, a system can verify the authenticity and integrity of the software update before installing it in the ECU. This closes the door to installing

malware updates which could put the entire system at huge risk by running malicious software or firmware.

In this thesis, we consider the use of secure OTA update framework during the update process of any software component as we will see in Chapter 5.

2.3.2 Detection Mechanisms

Despite the adoption of all the proactive security mechanisms mentioned in the previous section, there is no guarantee that the system becomes secure and that no attacker can strike it. These mechanisms are deployed and implemented to make attacks reasonably difficult, but they do not render them impossible (although they aim to); many examples of successful attacks against vehicles as a whole have proven that. Therefore, another layer of protection becomes necessary. An Intrusion Detection System (IDS) is a well-known security measure which is used to monitor the system to detect suspicious activities [MC14].

Intrusion Detection Systems Types

Based on the source of the collected data as well as the location where the IDS is installed, IDSs can be grouped into [DDW99]:

- **Host-based IDS:** which is used to monitor gathered data provided by the operating system from a single host to detect suspicious actions. This monitored data may contain memory consumption [VH17], the execution time of different software components which are mapped to that host [Zim+10; Ham+18], system calls [MP16], and so forth. SAMHAIN is a typical example of a host-based IDSs [Wot05].

- **Network-based IDS:** which is used to monitor traffic in a network environment to detect attacks. It needs to be installed at a point where it can monitor all the data transferred by the network hosts. Snort [CB04] is a good example of this type of IDSs.

- **Hybrid IDS:** a combination of both approaches which can be used to analyze the data within multi-host networked systems. Distributed Intrusion Detection System (DIDS) is one proposal for using host and network based IDSs together [Sna+97].

Detection Methods

All these types of IDSs can use different methods to detect attacks [Deb00; GT+09]. Here are the two main methods:

- **Knowledge-based intrusion detection:** known also as the signature-based [SM07] or misuse-based [BCV01] method. This method uses predefined rules based on accumulated knowledge about known attacks to identify any incidents which exploit these rules. This method has

many advantages; it is very effective for detecting known attacks. In addition, it has a very low false positive rate regarding detected attacks. But, Knowledge-based methods are not fully efficient at detecting zero-day attacks [GT+09; BD12].

- **Behavioral-based intrusion detection:** which is also known as an anomaly-based method [And+94]. By adopting this method, any off-nominal behavior (which is probably an attack) is detected based on the inconsistencies of the observed behavior with an off-line defined model which represents the system's nominal behavior. Compared to the knowledge-based mechanism, the behavioral-based method has better capability to detect unknown vulnerabilities and zero-day attacks [Had+12]. The main drawback of this method is the high false positive alarm rate of detected attacks. False positive alarms mean that there is a possibility that the system could report some activities as malicious although they are not.

Vehicular IDS

The automotive domain is different from the other traditional computer network systems; it has its distinct characteristics and limitations (e.g., low processing power) which may limit the direct adoption of existing IDSs from other domains. A large number of research studies have investigated using both types of IDS in the automotive domain, and using different detection methods [AJ+19; HL18; Wu+19; TBS18; Lou+19].

Host-Based IDS: most of the intrusion detection efforts in the automotive domain were in the direction of deploying network-based IDS for the in-vehicle network. Host-based solutions were very few. This could be a result of the limited resources of the hosts within the vehicle (i.e., ECUs). Also, the network tends to be the most attractive part of the automotive system because of the lack of security protection mechanisms, and thus it has received more attention.

The first approach which was used to develop host-based IDS for the modern vehicle is based on the execution behavior of the software components running on that ECU. Typically, each one of these components is implemented to follow a specific sequences of calls. The change in these sequences happens when the software is infected and tries to perform unusual or malicious actions. Such components can be detected by monitoring the execution sequence of each one and determining any variations compared to the nominal one. Based on this idea, Chen et al. [Che+07] proposed IDS to monitor software component behavior. They created a look-up table including all the possible predecessor/successor relationships of the monitored runnables. Later, they used this table to compare the actual execution sequence with the predefined one.

Another approach to detecting the off-nominal behaviors within each ECU is to look at its communication behavior. Larson et al. [LNJ08] followed this approach. They proposed building a model representing the nominal

behavior specifications for each ECU. These specifications contain the set of allowed out/in-going message types and the rates at which each of these messages are transmitted and received. In the same direction, other research (such as [LF12; AMS16]) proposed creating a list of all legitimate CAN message IDs that each ECU is authorized to receive and send, and later used this list to detect messages that are not predefined and treat them as malicious injected messages.

In this thesis, we proposed a host-based IDS to detect the malicious activity of software components based on their temporal behavior (see Chapter 6).

Network-Based IDS: The literature includes plenty of research which investigated and proposed different approaches for network-based IDS for intra-vehicle networks. The majority of these approaches were anomaly-based and designed for CAN bus system. This focus on the CAN bus is because it is used to control the most critical vehicle components, such as braking and engine control; thus, it needs to be especially well protected. Anomaly-based methods were preferred over signature-based ones because they require less memory to be developed (no need for storing attacks signatures) [Han+14]. Moreover, adopting the signature-based method requires frequent updates to include the newly discovered attacks and keep the attack signatures up-to-date during the whole lifetime of the vehicle. This imposes the need to visit the dealership whenever a new attack is found. However, the introduction of the OTA update technology will gradually diminish the impact of this factor.

Based on the typical behavior of the in-vehicle bus systems, Müter et al. [MGF10] have proposed eight attack detection sensors which could be used to observe anomalies occurring inside the vehicular network. These sensors include the formality, location, range, frequency, correlation, protocol, plausibility and consistency of each transmitted message. For this thesis, we focus on the approaches which form the nominal behavior model of the in-vehicle network based on the frequency and time analysis of the transmitted messages.

Most CAN messages are transmitted predictably at fixed intervals. This compatibility gave authors such as [TJL15; SKK16; HKD11; MV14], and others the ability to design IDS which use the frequency of each message (each message has a unique ID) to model its nominal behavior. During the operation of the vehicle, any malicious injected message will be noticed because it will have a shorthand time interval compared to the computed established baseline.

Miller and Valasek [MV14] were the first authors who developed a prototype IDS device to detect an injection attack on the CAN bus based on the frequency analysis of the messages. Their proposed device needs to be connected to the vehicle via the OBD port to monitor the CAN bus. In the same vein, Song et al. [SKK16] use message frequency to detect injection attacks as well as DoS attacks. Within their proposed IDS, they calculated the interval

between every two consecutive messages and compared it with the predefined one (during the learning phase). A short detected interval indicates an injection attack. At the same time, they monitor whether this short interval continues to appear in a row more than a preset threshold score; if yes, the model considers that the system is under DoS attack.

In the case of one ECU needs a certain data from another one. This ECU broadcasts a request (it is called remote frame) with particular ID, any ECU which is responsible for providing this data broadcasts immediately a response (data frame), which contain the requested data. Lee at al. [LJK17] proposed a mechanism to detect a message injection attack based on the calculation of the time interval between request and response frames for a particular message identifier when these messages are transmitted in an attack-free state over the CAN bus. The mechanism uses this computed time interval to compare it with the detected one during the run-time. If the detected time is different than the computed one, the mechanism considers that as a sign of intrusion.

If an ECU needs certain data from another one, this ECU broadcasts a request (called remote frame) with a particular ID, and any ECU which is responsible for providing this data immediately broadcasts a response (data frame) which contains the requested data. Lee at al. [LJK17] proposed a mechanism to detect message injection attacks based on the calculation of the time interval between request and response frames for a particular message identifier when these messages are transmitted in an attack-free state over the CAN bus. The mechanism uses this computed time interval to compare it with the run-time of the detected message. If the detected time is different than the computed one, the mechanism considers that a sign of intrusion.

All the proposed solutions focused only on the time-triggered (periodic) frames while the event-triggered ones were not considered.

2.3.3 Response Mechanisms

Detecting an intrusion using IDS in not sufficient to ensure the safety and security of the system; one or more reactions need to take place to respond to the detected attack. The IRS is designed to develop such reactions, handle the perceived malicious and suspicious actions of the attacker, and minimize the damage to the system. Most of the existing IRSs are mainly deployed for normal network systems and computer systems [Ste+01; Her17; Ina+16; Foo+08]. Based on the infected system, one or more of the IRSs can be implemented and selected.

In the vehicular domain, many authors have mentioned the need for an effective intrusion response mechanism for the vehicle system [FL15b; Sch12]. However, very few authors have investigated the design of such a system; in existing proposals, authors have looked at responses as a part of the intrusion detection framework [HKD08; FL15a].

A research study by Hoppe et al. [HKD08] was one of first attempts to develop a dynamic response system to react to the security incidents which target automotive systems. In their work, Hoppe et al. proposed the use of

the multimedia system to alert the driver to the detected incident. However, Informing the driver may not solve the issue, as the driver is not anything like a network (or network security) admin. On the contrary, it may cause the loss of the driver's attention in case of the massive number of alarms.

Other reactions include shutting down the communication bus whenever an incident is detected [MV14]. however, such a response may be too late as the attacker may have already gained a foothold in the local system. Also, the shutdown of the infected resources could be the target of the attack.

Recently, Vőlep et al. [VEV18] have proposed an intrusion-tolerant architecture to tolerate partial compromise of software components in an autonomous vehicle. Their solution is based on replicating the critical components to hide the actions of a minority of compromised ones behind a majority of healthy replicas operating in consensus. Even in the case of using replicas, whenever the attacker has a foot in the system (she was able to compromise one of the components), she may start attacking the other replicas and compromise the majority of them. In other cases, e.g., [NH14], an intrusion detection and adaptive response mechanism was designed to detect a range of attacks and to provide an effective response for Mobile Ad-hoc Networks (MANETs).

In Chapter 7 of this thesis, we propose an intrusion response system for vehicular systems which tries to address the issues of the currently proposed solutions and build upon them.

Honeypots

A Honeypot is a system which is used to mimic the target of attack. Although honeypots are considered as a detection mechanism by many researchers, they can be adopted as a reaction mechanism to malicious activities in the system. Honeypot is used to provide the attacker with a fake chance to perform its malicious activity. At the same time, it gives the system administrator a real opportunity to gather as much information as possible about the attacker and their goal. Eisenbarth et al. [Eis+08] have proposed the use of honeypots as a means of collecting such information about attacks against the in-vehicle network. In Section 6.1 of this thesis, we propose a mechanism which considers any infected component as a honeypot.

2.4 Summary

Over the past two decades, significant developments were introduced within the vehicular domain evolving the modern vehicle into a network of dozen ECUs hosting a huge number of applications. In Section 2.1, we presented an overview of the automotive system by describing its main components; namely ECUs which form the backbone of the vehicle, the software which implement the different functionalities, and the network which ensures the environment for these applications and ECUs to exchange information and collaborate to control the vehicle.

Additionally, in Section 2.2, we highlighted some security vulnerabilities which threaten each of the aforementioned components and the possible attacks they could enable. In Section 2.3, we summarized the mitigation mechanisms which were proposed for securing the in-vehicular systems. These mechanisms were classified into three groups: prevention, detection, and response mechanisms. Within each group we tried to cover the most relevant efforts for our proposed solutions.

"When the battle drum beats,
it is too late to sharpen your sword."
Winston Churchill

3 Environment

In this chapter, we explain our system architecture and the most critical functional and nonfunctional requirements (or goals) which need to be considered or achieved through this thesis. In addition, we present in Section 3.1 the hardware and software setup which is used to support the development and testing of the proposed security mechanisms. We also describe the use cases (in Section 3.2) which are used throughout the thesis to apply the proposed solutions and to show their applicability. At the end, to give the reader a general perception of our solutions, we briefly introduce our proposed multi-layer framework in Section 3.3. Each layer is discussed in more detail in a separate chapter in the next part of this thesis.

3.1 System Architecture

Figure 3.1 depicts an exemplary distributed system and the different scenarios of inter-component communication. In our idealistic world, each ECU is running an RTE that hosts multiple (interacting) software components. These components are safety and non-safety critical applications. The ECUs are inter-connected using IP-based Ethernet communications. Although we consider the architecture of a research vehicle, IP-based inter ECU communication has been proposed for automotive purposes (e.g., [Gla+10; Ker12; Bou+12]), so the chosen setup can be considered relevant for future automotive applications.

Here, we identify two types of network communication: *external* communication, which refers to the communication between one component within the vehicle and another one from the outside world (e.g., a component in another vehicle or with a road infrastructure). The second type is *intra-vehicle* communication which refers to the communication between two components belonging to the same vehicle. This type can be divided into another two types: *local* communication, which refers to the communication

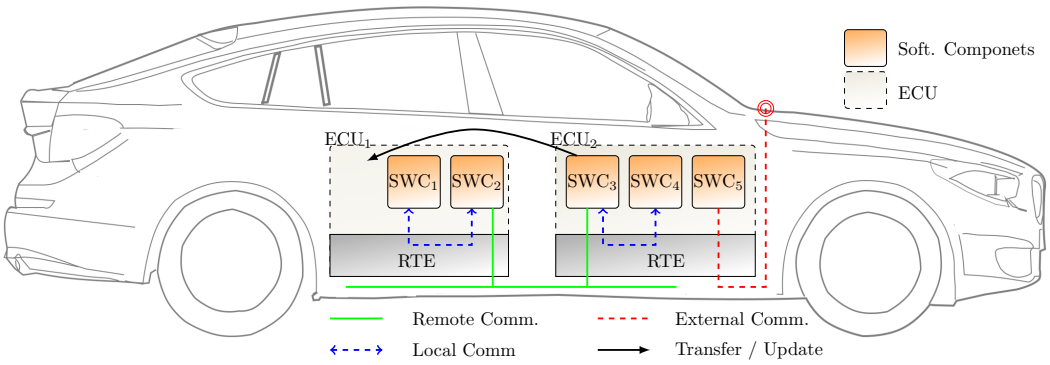


FIGURE 3.1: The different types of communication within in-vehicle systems

between two software components on the same ECU (e.g., SWC_3 and SWC_4), and *remote* communication, which denotes the communication between components on two different ECUs (e.g., SWC_2 and SWC_3).

3.1.1 Objectives

When it comes to designing a secure framework for distributed systems in critical application domains such as automotive systems, some objectives and requirements need to be considered. In the following we present the most relevant ones which should be respected:

- **Fine-grained access control:** The primary goal which the secure framework needs to guarantee is controlling *who should talk to whom* in an efficient manner without the need for static access-control mechanisms. Components should be isolated from each other. They should only communicate with other components which are specified by the security policy. One main aim of this isolation and restricted interaction is limiting the escalation of attack, so that even a compromised component will only be able to interact with authorized components and will be unable to attack other components indiscriminately.

- **Secure communication:** After specifying the authorized connections, providing security services for them is another fundamental requirement. The required security services vary from one application to another. Generally, most security issues in vehicle communications are related to the lack of data and origin integrity mechanisms; providing such mechanisms is necessary to prevent unauthorized parties from sending false data or injecting it in established connections. Thus, the proposed designed framework should ensure provision of these services.

- **Controlling Concurrent Change / updatability:** Nowadays, in-field updates are becoming ever more popular, as demonstrated by the recent OTA update for Tesla cars and others. New components could

be added or new configurations for a certain component can be altered. Any proposed security framework should ensure the integration of new components without putting the vehicle at risk of violating the system requirements or creating a vulnerability regarding communication and access control rules.

- **Composability and migratability:** In a component-based system where the functionality is integrated by composing several interacting components, we have the freedom of choice as to where to execute each component. Figure 3.1 illustrates this for component SWC_3 which could alternatively be executed on ECU_1 but then requires a remote communication mechanism to SWC_4. Hence, composability and migratability are important values whose absence would quickly restrict the design space. Note that we consider migration in terms of a (partial) system reconfiguration that must undergo several admission tests before being applied. Similar to the case of updating the system with new components, the security framework should support the secure composability and migratability of the components through a mechanism to develop and adjust the security policy to reflect the system reconfiguration without losing system security.

- **Efficient and Effective:** Another essential concern is the type of security protection that we must use for the various communications links. Applying encryption for all transmitted messages in the in-vehicle system could result in the opposite outcome by introducing an unacceptable overhead, which causes some functionalities to perform inappropriately. The resource-constrained nature of the hardware platform used, as well as the time-critical nature of some of these communications, imply that we need to be more selective in the type of protection we apply to these links. The proposed framework should provide efficient mechanisms suitable for resource-constrained devices. Also, it should support the effective selection of security properties.

- **Stability:** One critical mission of any proposed security framework for the vehicle is to implement different strategies to react to an attack and prevent the failure of the car. The response to a security violation must consider issues such as containment, continued availability, interaction with other subsystems, and, in some instances, latency requirements. These requirements are in many cases in conflict with each other; for example, security and availability are often in conflict [TGK00]. The security framework tries to react to a malicious component by stopping it. However, this means losing the availability of this component, which could cause a safety issue. Therefore, the proposed framework should consider such conflict and should attempt to resolve it or at least minimize it.

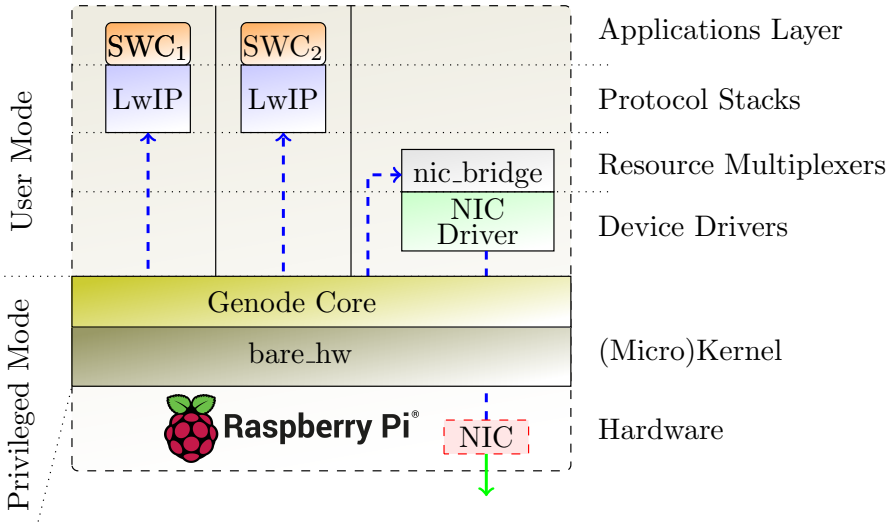


FIGURE 3.2: Hardware and software setup.

3.1.2 Prototype System

Figure 3.2 represents the hardware and software setup which is used in this thesis. This setup contains:

Hardware

We used Raspberry Pi as a simulator of a vehicle's ECU. Raspberry Pi is considered a resource-constrained devices similar to the most in-use ECUs in the vehicle. Also, Raspberry Pi has many other attractive properties; it is a small size computer board composed of an ARM-like System on Chip (SoC). It includes all the I/O and storage interfaces, a slot for a SD-card, and an Ethernet port. And, with all these properties it can be purchased at a low price (approximately tens of Euros). Moreover, it has a huge community of support. All this made Raspberry Pi a good candidate to test the applicability of our proposed solutions. It is worth mentioning that we are not the only people who used Raspberry Pi in the place of an ECU; another researchers have already done so (e.g., [Vau15]). In our test-bed, we have used a Raspberry Pi 2 Model B which comes with 900MHz quad-core ARM Cortex-A7 CPU and 1GB of RAM.

Software components

We used Genode framework [Gen19] as an RTE to develop our security solutions. The Genode framework can be seen as a collection of three layers: (1) The core component which implements only required system functionalities and gives the other component the ability to access raw physical resources such as Memory, CPU, and so forth. The core component usually runs on the top of a third party (micro)kernel in the user mode. Genode framework can be deployed over different kernels such as NOVA, seL4, Linux kernel (mainly to support rapid development and testing), and many others. In addition, it

can run directly on ARM-based hardware platforms (in our case this is Raspberry Pi 2 Model B) without relying on a third-party kernel. In this case only the core component runs in privileged mode. (2) The Genode components in the user space; these components contain the devices drivers, resource multiplexer and protocol stacks and (3) on top is located the applications layer.

Using a microkernels OS is the first step towards providing security for in-vehicle systems. A microkernel OS comes with a small-sized privileged code, ensures a minimal trusted computing base, and supports the memory protection between the various platform subsystems. Genode aims to create a generic user-level infrastructure regarding the microkernel it uses. Genode runs each application in a dedicated sandbox and isolated from the other components. In addition, each application has access to the rights and resources that are required to fulfill its specific purpose. Each process is exposed to only those parts of the system on which it ultimately depends. Thus, if one part of the system is compromised, the defect is limited to that particular part and its dependent parts, while unrelated processes remain unaffected.

Network

Genode relies on lightweight Internet Protocol (LwIP) [Dun02] as a TCP/IP stack to connect the applications to an Ethernet driver. Genode ported lwlP in the form of a library running in the user space and shares the same protection domain with the network application. Basically, lwlP implementation is focused on reducing the usage of memory resources and the code size to make it suitable for the embedded systems. LwIP does not include any IPsec implementation to support the secure communications between the different nodes which run Genode.

Genode provides a kind of network isolation by letting each network application have its own network stack. Besides, both the network stack and the network driver run in separate protection domains. In doing so, the system guarantees that unrelated components will not be affected by the faulty operation of the network stack. However, having separate network stacks for each application may introduce some issues, as we will discuss in Chapter 5.

3.2 Use Cases

Within the scope of the thesis, we consider the research vehicle MOBILE (see Figure 3.3-left), built at the Institute of Control Engineering [Ber14] to be used as a demonstrator platform with which we can describe and check the applicability of our different proposed mechanisms throughout the next chapters. MOBILE is equipped with multiple sensors and computers for environment perception with the goal of providing obstacle avoidance and Adaptive Cruise Control functionalities in a static environment on a proving ground.

3.2.1 Autonomous Vehicle Driving

Self-driving vehicles represent the car of the future and the main goal of all automotive manufactures. During the last few years, huge efforts have been made to develop self-driving vehicle technology. Such efforts were pushed by the rapid progress achieved in the hardware (e.g., cameras, sensors, etc.) and technologies areas related to autonomous driving (e.g., artificial intelligence, big data, etc.). However, there is still a long way to go and more work needed to develop the fully autonomous vehicle (i.e., SAE Level 5 [Com+14]).

Figure 3.3-right depicts the simplified hardware architecture of the autonomous vehicle driving system as well as the network architecture for the main subsystems. Three types of hardware components are used within this use case: a) **Smart sensors** (e.g., GPS, LiDAR, etc.) which are used to gather data about the environment and deliver it to b) Many **ECUs** running the different autonomous vehicle software systems which emit control commands to the c) **Actuators** (e.g., steering, throttle, brakes, etc.) which apply those commands.

The autonomous vehicle software components can be categorized into four main subsystems [Pen+17; Cam+10]:

- **Localization:** refers to the ability of the vehicle to determine its position concerning the surrounding environment as well as to get a good estimation of road traveled. This position information is fed by GPS and inertial data via CAN bus to the ECU which is responsible for vehicle localization and motion estimation.

- **Perception:** translates the received raw sensor data about the surrounding environment into useful information to obtain a safe trajectory. Lidar sensors and a camera are streamed via IP-based protocol to the ECU which is responsible for environment perception (sensor data processing, data fusion, environment modeling). Data from a radar sensor is acquired via a CAN bus connection.

- **Planning:** the main aim of this subsystem is utilizing aggregated data, which is provided by the perception subsystems to plan actuation of the vehicle. This includes the optimal trajectory planning, behavioral planning, motion planning, etc. The planner unit passes the ultimate information/commands to the control unit.

- **Controlling:** this subsystem receives the ultimate information/commands of the planner unit and passes them to the actuators which generate the desired actions such as increasing the speed or moving the steering and so forth.

Next, we introduce the two use cases which are employed in the next part of the thesis:

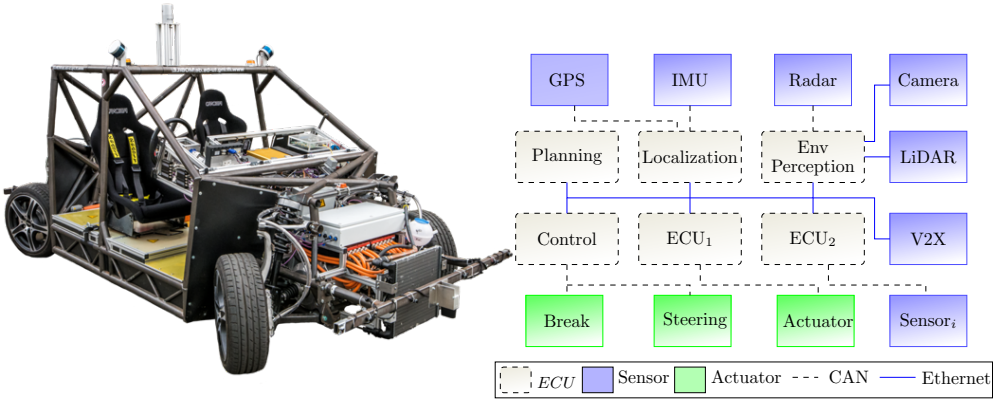


FIGURE 3.3: MOBILE (left) and a simplified hardware architecture and software components for autonomous driving system (right)

3.2.2 Automated Obstacle Avoidance

This is one of the vital features which aims to ensure the safe vehicle operation and ride comfort of the autonomous vehicle. This system enables the car (with the help of the information received from the different sensors) to avoid obstacles in its path. The environment perception which is used by this system is mainly based on LiDAR scanners (based on [RMM15]); a radar sensor and camera present additional information sources which can be used to monitor the environment around the car. The LiDAR sensor data is used to create a map of the static environment, which provides the basis for model-based trajectory planning to generate smooth maneuvers in real-time.

3.2.3 Adaptive Cruise Control (ACC)

ACC is a common and well-known automotive driver assistance system which has been developed by most of the automobile manufacturers, which give it different names, for example BMW's Traffic Jam Assistant, General Motor's Super Cruis, Tesla's Traffic-Aware Cruise Control, Toyota's Automated Highway Driving Assist, and many others [DW+14].

The cruise control system, which keeps the vehicle at a steady speed entered by the driver without any need to keep the accelerator pedal continuously pressed down, was integrated with the vehicle a long time ago. The ACC system is considered an improved version of this old system. An ACC system supports the driver to adjust the vehicle's velocity and to maintain a safe headway distance between vehicles in the same lane. The ACC System depends on user input to determine the desired speed or distance with the in-front car. Internal sensors such as LiDAR and a camera are used to provide information about the speed of the car ahead which helps the ACC system to adjust its distance from it by controlling the engine and brakes.

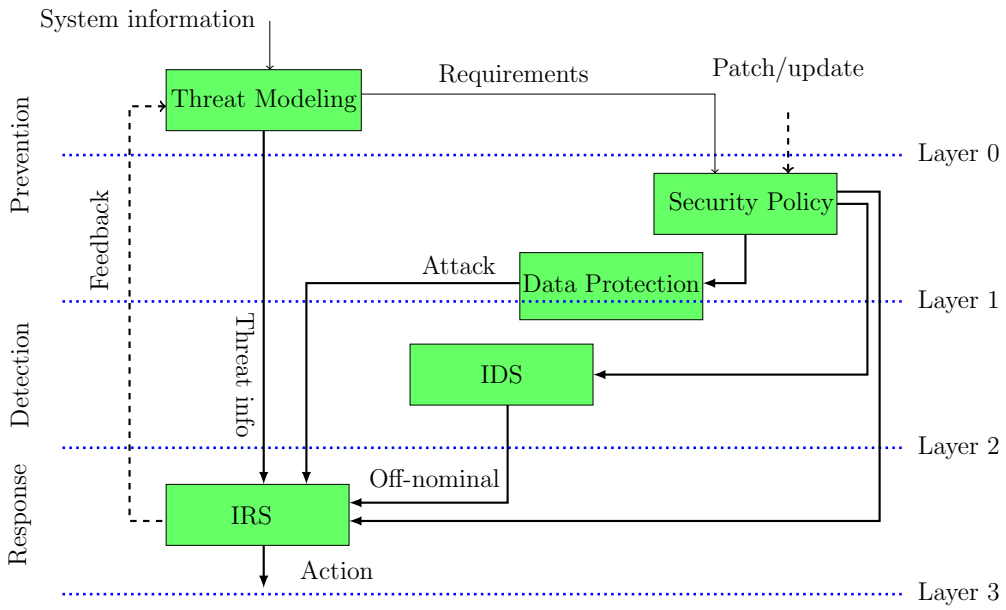


FIGURE 3.4: The different layers of the proposed secure framework

3.3 Multilayer Vehicle Security Framework

In order to ensure the security of the in-vehicle system we propose a holistic framework, shown in Figure 3.4. This framework consists of multiple layers of mitigation strategies to deal with cybersecurity intrusions and to ensure a high degree of security for the intra-vehicle systems.

Our defined framework is based on the cybersecurity framework proposed by the National Institute of Standards and Technology (NIST) [Adm+16; Cyb14]. The NIST framework uses five principal functions "Identify, Protect, Detect, Respond, and Recover," to build a comprehensive approach to developing layered cybersecurity.

3.3.1 Design

Our framework depends on three principles, namely: prevention, detection and response, which cover most of the NIST framework's functions.

- **Prevention**: which is covered by the first two layers of the framework. Within the first layer (i.e., **layer 0**), we perform a comprehensive thread-molding process to identify all the possible threats and vulnerabilities which could face the system. This process requires a good understanding of the system functionalities as well as the system components and their interconnections. One main outcome of this process is defining the security requirements [MLY05] for the different system assets. These requirements need to be satisfied to prevent attackers from compromising the system. Chapter 4 discusses this layer in more detail.

The second layer (i.e., **Layer 1**) includes developing the security policy which implements the security requirements for each system component. The security policy contains rules which will be used by the other layers of the framework. Another process within this layer is defining the mechanism which is used to enforce the security policy as well as to implement a mechanism to ensure the security requirements. This function includes building an access control mechanism to prevent unauthorized parties from accessing the system assets. Also, it adopts some crypto mechanisms to ensure data security. Some of these mechanisms, such as a firewall, can be considered as detection and prevention tolls at the same time, therefore can overlay on the next layer. Chapter 5 describes this layer.

- **Detection**: The core function of this layer (i.e., **layer 2**) is building monitoring mechanisms to detect the occurrence of cyber-attacks which were not prevented by the prevention layers. This layer ensures continuance monitoring as well as the early detection of attacks and suspicious actions. Both anomaly-based and signature-based detection technologies need to be adopted in this layer. Security policy plays a pivotal role in defining the nominal behavior of the system components. This defined behavior is used as a reference to detect any odd behaviors. Chapter 6 of this thesis presents a proposed intrusion detection mechanism for in-vehicle systems.

- **Response**: The last layer (i.e., **layer 3**) determines the process when a security violation is detected either by layer 1 or layer 2. The main aim of this layer is protecting the vehicle from entering unstable states as a result of the detected attack. The effective design of this layer requires inputs from the above layers, such as the properties of the detected attack or the off-nominal behavior. The security policy is also needed for this layer since it includes the possible response for different attacks.

  Besides that, the collocated characteristics of attacks (i.e., aim, scenario, severity, etc.) are used as feedback or input for the next threat modeling process. In addition, it can be used to develop patches or updates for the security policy to mitigate these attacks. This update will be reflected on the underlying layers. Chapter 7 handles the development of this layer.

3.3.2 Development and Usage

The development of our multilayer framework is achieved gradually over the different security phases of the automotive system development lifecycle. We follow the security phases which were presented in [Bur+12] and recently proposed in the first working draft of the ISO/SAE 21434 standard. These phases, which are designed similarly to the safety stages defined in ISO 26262 [Isoi], aim to provide guidelines for designing secure systems. Figure 3.5 shows these phases, which include defining security requirements and goals, security design, security implementation, security testing and

verification, security validation, and security maintenance. The same figure shows how our proposed framework layers can be combined into these phases.

The development of layer 0 takes place off-line at the lab during the conceptual phase. This layer is used to derive the security requirements of each (sub)system component. In addition, the attack scenarios which are produced by this layer are used as realistic use cases for penetration tests during the security validation phase. The layer 1 mechanisms (i.e., security policy and data protection protocols) are developed as a part of the system security design, implementation, and integration. The security policy will be used later while the vehicle is operating to support the security monitoring process via developed monitoring techniques. The mechanisms in layer 3 and 4 are used as part of the security maintenance and monitoring phase while the vehicle is in the field (i.e., after production). Layer 4 information about threats is used later as part of the security vulnerabilities handling and security updates.

In the next part of the thesis, we have dedicated a chapter to each of these layers to discuss them in more detail.

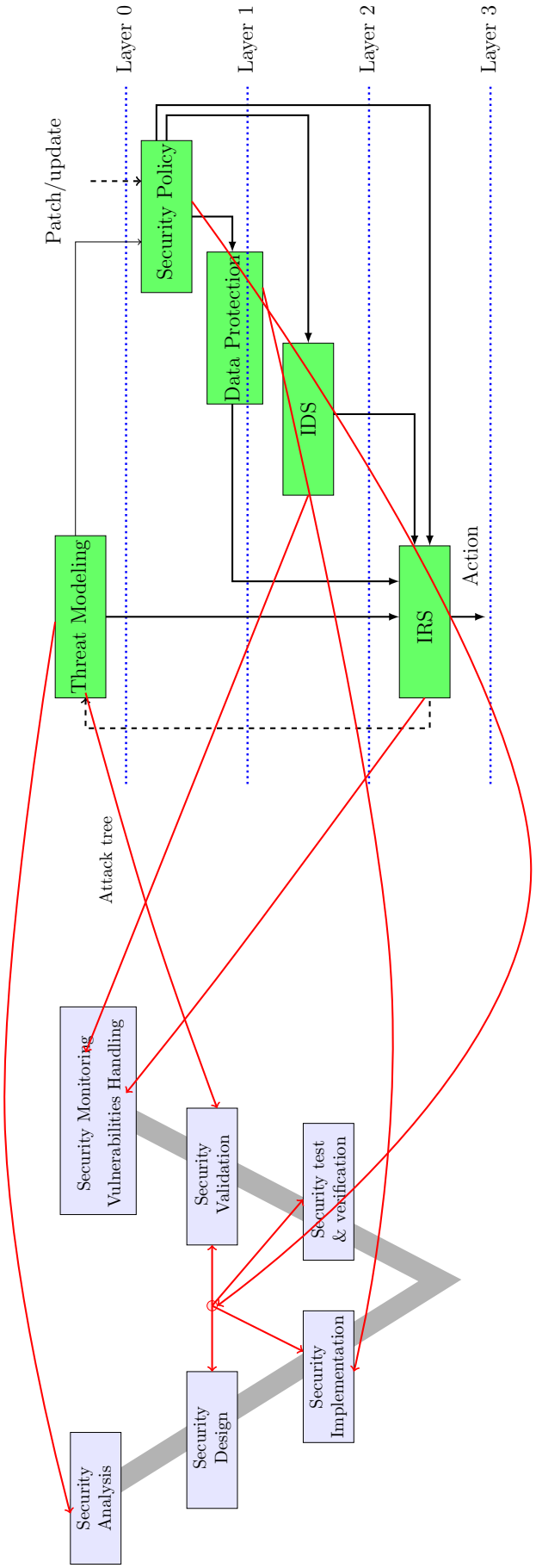


FIGURE 3.5: The mapping between our multilayer framework and the different vehicle security development phases

Part II

SOLUTIONS

"If you know the enemy and know yourself you need not fear the results of a hundred battles"

SUN TZU

4 Comprehensive Threat Model

The increase in connectivity within vehicles is a double-edged sword. On the one hand, it extends the vehicle's functionalities and capabilities, but on the other hand, it opens the door to several cybersecurity threats and makes the vehicle a more attractive target for adversaries. The safety critical nature of the vehicle requires the adoption of high security measures when developing vehicular IT systems. A good understanding of security requirements, which can be concluded from threat modeling, is a primary step towards contriving sufficient security countermeasures. Threat modeling helps to identify and address most of the potential threats. In fact, threat identification would likely reduce the life cycle as well as the cost of achieving security objectives when it is considered during the design process. Furthermore, threat modeling provides relevant information about the attack vectors which threaten the system. Such information can be used as a reference during the test process to avoid the omitted threats.

Different researchers scrutinize threat modeling in the vehicular domain. However, most of the research examines the potential threats only partially by focusing on a certain aspect or by looking at threats which affect a particular sub-system. Practically speaking, the lack of a general threat model within the vehicular domain makes threat analysis for the different subsystems a resource-consuming task. Additionally, it increases the possibility of inconsistencies between the interacting subsystems and causes redundancy when defining the attack vectors.

In this chapter, we revise the existing vehicle-related threat modeling efforts to develop a comprehensive threat model for the automotive domain. Our model seeks to combine multiple approaches to arrive at a more comprehensive one. Within our model, we define various potential groups of attackers, the nature of the attack, potential targets, and the security requirements for the vehicular domain. Then, we propose an abstract model which can be used to classify all conceivable attacks against the vehicular domain.

The abstract model is used as an aid to construct general attack trees [Sch99] which illustrate attack vectors that threaten a particular vehicle sub-system. Parts of this work have been published in [HNP16].

The rest of the chapter is organized as follows: In section 4.1, we review existing threat models in many IT domains including those which were proposed for use with the vehicular domain. Then, we present our proposed threat model in section 4.2 and explain its components. Section 4.2.6 presents a general model to identify possible threats within the vehicle. In Section 4.4, we used our general model to identify threats within an automated obstacle avoidance use case. Finally, we present our conclusion in section 4.5.

4.1 Threat Modeling

Before we can explain what a threat model is, it is necessary to define some basic principles that we will need.

Definition 4.1.1. ***Asset:*** *"any system resource which needs to be protected" [Shi07b].*

Definition 4.1.2. ***Vulnerability:*** *"A flaw or weakness in a system's design, implementation, or operation and management that could be exploited to violate the system's security policy" [Shi07b]*

Definition 4.1.3. ***Threat:*** *"A possible danger that might exploit a vulnerability" [Shi07b]*

Definition 4.1.4. ***Attack:*** *Any malicious activity that attempts to collect, expose, alter, disrupt, disable, degrade, destroy or gain unauthorized access to components or the data exchanged during the communication by exploiting one or more vulnerabilities [Isob].*

Based on the previous definitions, we next provide the definition of threat modeling:

Definition 4.1.5. ***Threat Modeling:*** *A systematic process used to (a) analyze the potential* ***attackers****, (b) identify the security* ***vulnerabilities*** *and* ***threats*** *in the (sub-)system* ***assets*** *which could be exploited by an attacker and (c) provide significant information that would help to safeguard the target (sub-)system against attacks as well as to develop realistic and meaningful* ***security requirements*** *for this (sub-)system.*

Different approaches were used to develop threat modeling. Each of these approaches focuses on a certain aspect on the system such as asset, attacker, software, or system vulnerabilities [Sho08; Sho14]. Typically, threat modeling has been implemented using one of these different approaches independently. However, using more than one approach at a time has recently started to become more common [Mea+18; Eng17]. Next we will look at each of these approaches in turn and outline how they have been adopted in the automotive domain.

4.1.1 Attacker-Centric

This approach focuses on profiling attackers' characteristics, goals, skills, and motivations for targeting the studied system. By putting ourselves in the shoes of the system attackers, we can understand their goals, which provides us with valuable information about the most targeted assets in our system. Consequently, this gives us a better understanding to implement appropriate mitigation strategies for stopping these attackers.

Intel Threat Agent Library (TAL) [Cas07] is one example of attacker-centric threat models. TAL standardized a list of 22 threat agents that pose threats to IT systems. The classification of the agent is based on eight unique attributes: the intent of the agent (whether to cause harm to the system or not), the way the agent can access the assets, the agent's main outcomes, the limits which constrain the agent, the resources available to that agent to carry out the attack, the skills that the attacker must have to target the system, the agent's objective, and the identity visibility of the threat agent. In addition, for each of these agents, the model keeps an updated rating based on many factors such as that agent's recent attacking activities.

Intel's Threat Agent Risk Assessment (TARA) [Ros09] is one of the main examples of threat modeling methodology which consider the threat agent as the origin of the security risk. TARA aims to narrow the field of all possible attacks against the system and determine the most probable attacks based on the TAL agent list and the rate which is given for each agent.

Within our previous work [HNP16], we were among the first authors to attempt to concentrate on the attacker-centric method for the automotive domain by classifying attackers against the vehicular system and trying to define their different goals.

Other efforts also followed the same strategy and included the attacker in their threat model, such as [Mon+18]. One of the most interesting efforts was by Karahasanovic et al. [KKA17]; the authors proposed the adaptation of the TARA threat model for the automotive system. To achieve that, they made some changes to TARA to make it fit with the automotive industry's needs.

4.1.2 Asset-Centric

This approach focuses on system assets to protect them from attackers. Assets, as stated in Definition. 4.1.1, can refer to anything valuable in the system; this includes people, software, hardware, data, and so forth. The idea behind this approach is that by profiling assets which are more attractive to attackers and identifying their vulnerabilities, the organization can wisely invest in protecting these assets and does not waste resources guarding non-critical or unattractive assets.

Operationally Critical Threat, Asset, and Vulnerability Evaluation (OCTAVE) [Car+07] is an example of asset-centric threat modeling methodology. This approach consists of four phases: during the first phase, the organization develops risk measurement criteria. In the second phase, the critical assets are profiled and their security requirements are defined. Identifying

the threats to these assets is accomplished within the third phase. Finally, analyzing the risks related to the different assets and starting the development of mitigation approaches take place during the fourth phase.

The Threat, Vulnerability, and Risk Analysis (TVRA) model, which was proposed by the European Telecommunications Standards Institute (ETSI), is one approach aims to support the security analysis in the vehicular domain by focusing on the assets of the system [ERS10]. The TVRA process starts by defining the security objectives and requirements of the system. Then, it defines a list containing all the system's assets is produced. For each asset in the defined list, classification of all possible vulnerabilities and related threats is carried out. The risks of threats are determined based on the likelihood of the identified threats. Finally, countermeasures are proposed to reduce the risks of the threats.

4.1.3 Vulnerability and Threat-Centric

Instead of considering all existing vulnerabilities in different assets, this approach aims to focus on the risky vulnerabilities which are most reachable and desirable for attackers and to develop mitigation for them. Usually such vulnerabilities exist within the assets that attackers can easily reach (i.e. attack surfaces or attack entry points).

Skybox Threat–Centric Vulnerability Management (TCVM) [Sky] is an example of such an approach. The TCVM process starts by profiling all existing vulnerabilities in the system's assets. The next step is identifying the exposed and exploitable vulnerabilities from the existing ones based on context-based prioritization and management techniques.

In the vehicular domain, many researchers have focused on identifying the vulnerabilities of the various vehicular system assets. Checkoway et al. [Che+11] have studied potential attack surfaces of the vehicle which could be exploited by attackers externally. On the other hand, Koscher et al. [Kos+10] investigated the attack surfaces on the underlying system structure. Both demonstrated that attackers could leverage direct access to the CAN bus to control various functions by exploiting vulnerabilities in the attack surface.

4.1.4 Software-Centric

This is also called design-centric. As the name suggests, this approach focuses on security during the design phase of software components for the system by identifying the threats which may expose each component. Loren Kohnfelder and Praerit Garg at Microsoft proposed a software-centric methodology for threat modeling called STRIDE [KG99], which stands for the major six attack categories which threaten software products: Spoofing of user identity, Tampering with data, Repudiation, Information Disclosure, Denial of Service and Elevation of Privilege.

Winsen [Win17] proposed using STRIDE to identify all possible threats for future autonomous and connected vehicles. Then, he analyses these threats based on their severity and controllability.

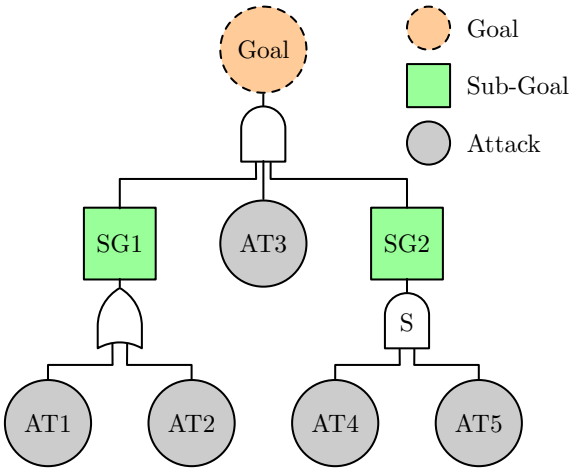


FIGURE 4.1: An Attack Tree

4.1.5 Attack Trees

Threat analysis describes *who* the potential attacker is, *what* are the motivations behind an attack are, and *which* components could be threatened. Describing *how* an attack could be executed is the mission of attack trees. An attack tree is used to explain attacks in a tree structure as shown in Figure 4.1. The root of the tree represents the attacker's ultimate goal, while the intermediate nodes of the tree (sub-goals) define different stages of the attack. In the case that a node in an attack tree requires all of its sub-goals to be achieved, the sub-goals are combined by an AND branch. If a node requires that any of its sub-goals be achieved, the sub-goals are combined by an OR branch. Leaves nodes represent atomic attacks. Attack scenarios are generated from the attack tree by traversing the tree in a depth-first method [MEL01]. Each attack scenario will contain the minimum combination of leaves. In classical attack tree models the attack chronology is disregarded. However, in many cases, the success of an attack depends on the subsequent success of interrelated attack steps. Arnold et al. [Arn+15] propose sequential AND- and OR-gates (SAND, SOR) to handle sequential occurrence of attacks.

Attack trees have been used as a tool to illustrate the attack steps for individual attack scenarios within the vehicular system [HNP16; Izo+16; NPR18]. Henniger et al. [Hen+09] proposed the use of an attack tree to identify attacks and formalize the security requirements for the automotive's on-board networks. Aijaz et al. [Aij+06] tried to create a reusable attack tree for V2V communication threats. In our work, we are attempting to provide an abstract model which helps to create a general attack tree for the entire vehicular domain.

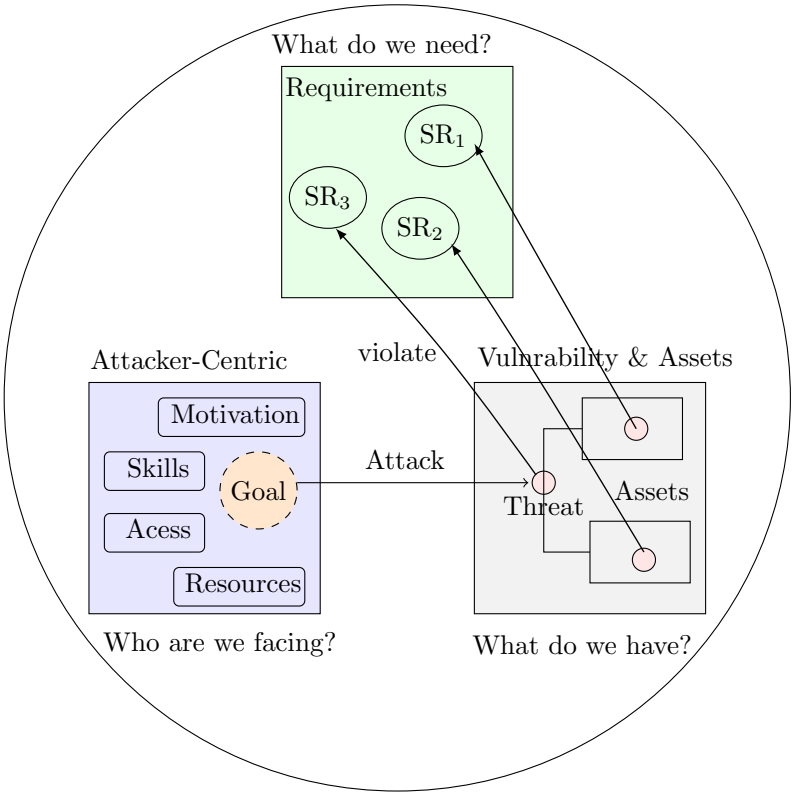


FIGURE 4.2: SAVTA threat model

4.2 SAVTA: A Comprehensive Vehicular Threat Model

As we showed in the previous section, there are many threat modeling approaches that have been implemented within the vehicular domain. Using one individual approach will not identify all the threats which could target the system and may lead to insufficient mitigation mechanisms [MHC14; Mea+18]. For example, applying an asset-centric method requires the definition of the most critical assets which an attacker may target. However, the type of attacker (whether she is internal such as the driver, or external) will affect the selection of the potential attack surfaces from these assets.

Effective defense against threats requires addressing all existing security flaws in the target system and identifying threats which exploit these vulnerabilities. In addition, it demands a good comprehension of the prospective attackers, their capabilities, and their objectives. To address all this, we propose a threat model called **SAVTA** (**S**oftware, **A**ssets,**V**ulnerability and **T**hreat, and **A**ttacker - centric method) which is a hybrid threat model that adopts different existing approaches, bringing them together to create a comprehensive threat model for the vehicular system.

Figure 4.2 shows a general view of SAVTA threat model and how the various approaches are interconnected with each other. This interaction reflects the threat model definition (i.e., Definition 4.1.5) which was given in the previous section. Within a very complex environment such as a vehicle,

different assets coexist. These assets usually suffer from several hidden vulnerabilities (as shown in Section 2.2). A motivated attacker could target each of these assets by generating suitable conditions to exploit one or more of these vulnerabilities. Exploiting any of these vulnerabilities always ends up with violation of one or more of the security requirements.

The next steps are required during application of the SAVTA threat model:

- We start the process by looking at *what are we facing?* we can address this question by defining the *Attacker Profile* which includes information about all possible attackers who may target the vehicular system. We classify these attackers based on their motivation and the other resources that they have or use to reach their goal.

- In addition, we have to answer the next main question *what do we have?* the answer to this question should cover all the possible weak points that we have in our system. We achieve that by identifying all the vehicle's *Attackable assets*; and for each of them, we have to identify all the vulnerabilities and threats which affect it. We also need to define the relations which interconnect the different assets. These relations are a reflection of the functional requirements that the system needs to work. By defining the relations between the different assets, we can point out the assets on which all others depend for accomplishing the objective (i.e., the centre of gravity [VCHP84; Ste+18]) as well as the possible attack chain which leads to defining all attack surfaces

- The last question we need to answer is *what do we need* to secure these assets? *Security requirements* need to be determined for each asset as well as the relationship between them. Determining these requirements involves defining the references for the security mitigation implementation against attacks that may threaten the assets, for example, the integrity requirement of the communication link which could be threatened by a replay attack.

- After obtaining the previous information (it is important to note that the three previous steps can be achieved in parallel and carried out by different security teams), we start linking them together to create an abstract model which can be used to classify and identify all the vulnerabilities, threats, and attacks.

Next, we discuss each of these steps in detail.

4.2.1 Attacker Profile

Different groups of attackers are attracted to attacking vehicles. These groups range from the owner of the car to an expert hacker with sophisticated tools. One of the most important steps towards securing the vehicular system is knowing the goal of threat agents (i.e., attackers) as well as other properties of each agent, such as motivation, skills, resources, and the accessibility of

the attacked object, which can play a significant role in reaching this goal. Understanding these properties for each attacker will enrich our knowledge and improve our capabilities to define adequate mitigation. We start defining the possible threat agents for the automotive domain by looking at the motivations behind the various attacks that we have faced:

- **Falsification:** An attacker (who could be the owner or the driver) may wish to misrepresent actual vehicle information such as changing the achograph or odograph measurements to sell the car with a false mileage reading or to defraud the operator and safety regulations [And98].
- **Illegal profit:** An attacker could make a profit by stealing the vehicle or by selling the attack capability to a different organization. Some attacks could be driven by a commercial competitor of the target vehicle's vendor to sabotage their product and gain market share. Although there is no published case which states that a particular attack has been carried out based on industrial espionage, such motivation remains valid since there is a history of industrial espionage between vehicle manufacturers [Mer97]. Such an attack requires the collusion of one or more insiders in the targeted organization.
- **Fun and vandalism:** Revenge and vandalism could motivate some attacks, as in the case of a dismissed employee who sought to punish his ex-company by bricking cars sold by this company [Pou10].
- **Research and test purposes:** Attacks and penetration tests could be performed by security experts or test teams. The attackers in this case have benign motivations. They seek to discover security flaws in different components of the vehicle systems before they can be exploited by third parties.
- **Accidental:** In some circumstances, an attack could happen without any intention as a result of lack of security knowledge or the lack of security tests being performed. A very well-known example of such an attack is the one which faced Toyota Lexus vehicles in 2016 [Bog16] and caused the GPS, climate control, front console radio systems USB, Bluetooth, and other features to stop working suddenly after an update to the Enform system. The company stated that the reason for that malfunction was an unintentional reading of errant data, which include traffic and weather data.
- **Terrorist:** Although there are still no real incidents in which such motivation has been proven to be behind the vehicle cyberattack; Richard Clarke, Former U.S. National Coordinator for Security Infrastructure Protection and Counterterrorism, hinted that the fatal crash of journalist Michael Hastings' Mercedes C250 coupe is consistent with a car cyberattack [Nim13].
- **Overlap:** Sometimes, multiple motives could lie behind a single attack.

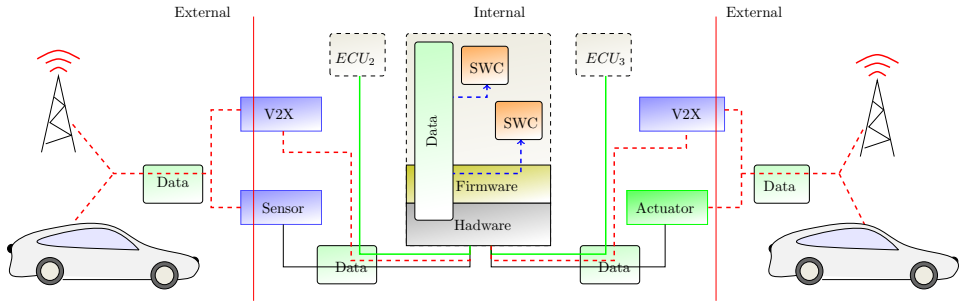


FIGURE 4.3: The different assets of the automotive system

It is important to note that this list is not complete; other motivations can be added whenever we discover a new attack. However, motivation alone is not enough. An attacker needs sufficient resources to achieve his goals. These resources include: skills, capabilities, technical equipment, and financial resources. The disparity in these resources could be used as a tool for adding another level of classification of the attackers.

- **Limited:** Attackers in this group have limited financial resources and insignificant knowledge about the vehicle architecture. Such attackers lack the ability to use complicated tools. Regular car thieves, owners who would like to install or replace a component within their cars, an attacker who tampers with highway signals to gain a reputation in their community, unsophisticated attackers (script kiddie) and others are good examples of members of this group.
- **Adequate:** This group includes highly skilled experts who have adequate tools and equipment to perform the attack. But, they may not have sufficient financial resources to support them or to build bigger teams and cause huge damage. However, the members of this group could use their experience to obtain profit, such as by operating as black-hat hackers. Mechanics, owners with good knowledge and security researchers belong to this group.
- **Sophisticated:** This group contains expert hackers with sophisticated tools and huge financial support. A good example of this group are the cybersecurity organizations (governmental and nongovernmental) who have multiple members of the above group who work together. Typically, massive financial support enables them to obtain the sophisticated tools and attract experts. Some security research groups with similar resources could be another example of this class.

4.2.2 Attackable Assets

Attackers may focus on different parts of the vehicle components. One of the main steps of the SAVTA method is to determine these assets and identify the vulnerabilities and attacks which threaten them. In autonomous vehicles (see section 3.2), a software component in one ECU depends on the information provided by a set of sensors or transmitted by other components in

another ECU to perform its function. This function could be translated into a physical action by different actuators or transmitted as an input for another component in another ECU. For such a system and based on definition of the asset (Definition 4.1.1), we can identify the next vehicle assets (see Figure 4.3) which may be targeted by the different threat agents and that need to be protected:

- **In-Vehicle Hardware:** This includes the different ECUs distributed across the vehicle. Reaching the in-vehicle hardware gives the attacker the opportunity to replace any legitimate device with a malicious one, or even to install new hardware which could cause havoc. Such hardware devices may not be part of the vehicle. They could be third-party devices plugged into the vehicle, such as the driver's mobile phone [Izo+16]. The smart sensors (e.g., camera, LiDAR, radar, etc.) within the vehicle can be another hardware targets to various attacks such as camera sensor blinding attacks [Pet+15], LiDAR blinding attacks [Shi+17], and others. These attacks aim to prevent these sensors from working properly, which has a serious safety impact on both autonomous and normal vehicles.

- **Software and Firmware:** The massive amount of integrated software in each vehicle and the different levels of security auditing between the different vendors make the software more susceptible to attacks. Attackers can benefit from software vulnerabilities to inject malicious code, causing software components to behave maliciously or to stop the application and prevent the vehicle from achieving certain functionalities. The firmware which controls the ECU could be a target for various attacks; some attackers could tamper with the ECU firmware to achieve superior performance [WA15]. A malicious update of one application or of internal parts of the firmware could open the door for the attacker to inflict damage to the vehicle.

- **Data:** The huge amount of data exchanged, the architecture of the in-vehicle network used to transfer this data, and the unsophisticated hardware (i.e., ECU) used to store it make data a very attractive asset for security attacks. Attackers can target *data stored* in some ECUs; this data could be the execution code of the applications, the firmware drivers, crypto private keys, digital certificates, or private vehicle and driver activities (e.g., vehicle location, navigation destination, etc.). Extracting such data may occur by targeting the hardware through side channel attacks, or by targeting one malicious software component or the OS itself, especially if that OS does not store the data properly with strict access restrictions. Alternatively, attackers could threaten *transmitted wired/wireless data* within the vehicle.

    a) In-vehicle data exchanged between different components or between one component and its sensors or actuators. Attacking this data depends on the existence of vulnerabilities in the internal network protocols. Spoofing, altering, or drooping the transferred

data between the on-board system and different sensors or actuators are examples of attacks against such data [Rou+10].

b) Data transferred between the vehicle and the external world; such as V2V communication data, V2I communication data, etc. This data could be targeted after it is received by the sensors; in this case, it is treated as in-vehicle exchanged data. In other cases, the data can be infected by different attacks (such as spoofing and emitting false data) before it has been revived by sensors [Pet+15]. Finally, the data could be targeted before it leaves its source.

- **Surrounding infrastructure:** Another critical asset which could be targeted by attacks is the surrounding environment of the vehicle. Although the surrounding environment, such as other vehicles, roadside units (RSUs), etc., is considered external to the in-vehicle system, it has a direct impact on the security of the in-vehicle system since it is the data source for sensors and V2X. Note that the under-study in-vehicle systems (or the vehicle as a unit) is considered external for other vehicles moving on the same road or for road infrastructure. Thus, the surrounding environment can be studied similarly to in-vehicle systems by looking at its threat agents as well as its assets (i.e., hardware, software and framework, data). Many security attacks can be launched against the surrounding infrastructure. A typical example of such an attack is adding stickers to traffic signs [Eyk+18]. Another example is modifications to electronic road signs, such as "Zombies Ahead", where an attacker figured out how to alter the text on electronic road signs to create warnings of a zombie attack. Even such a ridiculous attack could create public safety issues for drivers on the roadway [Olo14].

We need to define a list of all these assets for each subsystem within the vehicle. The definition of these assets depends on the security analysis and how it sees the system as well as how much information it has about the different subsystems. Assets can be defined in a very *abstract way* by looking at each functionality of the system as a black box which will be mapped in a hardware platform and will intercommunicate with other functionalities using very generic network architecture. From such a viewpoint, the transmitted data will reflect the functional relationship between the different functionalities. A *detailed overview* can be achieved if the information about the software components as well as the actual network architecture is available. This detailed information can be derived from the abstract view by decomposing each defined functionality into its actual software components, RTE components, devices drivers, protocols, etc. Many tools can be used to visually represent the different assets in the system in the different ways; one example of such tools could be flow diagrams (DFDs) [Sho14, p. 44-47]. Any other equivalent type of diagram can also be used. Figure 4.3 shows one representation of the different system assets.

The main point here is to distinguish between the local (e.g., functionalities, sensors, actuators inside the vehicle) and external domains (e.g., nearby

TABLE 4.1: Mapping STRIDE with CIA^3 security attributes.

STRIDE	Security attribute (CIA^3)	Explanation
Spoofing Identity	**A**uthenticity	pretend to be someone else
Tampering with Data	**I**ntegrity	improper assets alterations
Repudiation	Non-Repudiation	trackless
Information Disclosure	**C**onfidentiality	to access to confidential data
Denial of Service	**A**vailability	disable or delay accessing an asset
Elevation of Privilege	**A**uthorization	perform unauthorized actions

vehicles, RSUs, etc.). Functionalities (or components) which have relationships with external domains are usually more attractive targets and may represent reachable attack entry points for attackers because they are more accessible than others.

4.2.3 Attack Effects

The relationship between the different assets plays the main role in determining the effect of the attack on a specific component. For this contribution, we classify attacks based on their effect:

- **Limited attack:** The ultimate target of some attacks could be a single part of the vehicle. The effect of such attacks will remain limited to the attacked ECU(s) and not propagate any further. The targeted system will define the jeopardy level of the attack.
- **Stepping stone attack:** The attack can start by compromising one component or subsystem. Later, the attacker uses this subsystem as an attack surface to plague all related subsystems. The same process can be repeated for the newly infected components. Koscher et al. [Kos+10] showed that an attacker who can control one ECU is able to attack other connected ECUs.

4.2.4 Security Requirements

For each one of the defined assets, a set of security requirement need to be checked. These requirements are reflection of the various threat which may threaten that assets. Thus, for each asset, we check weather it suffers from one or more of the STRIDE model components. As we can see in Table 4.1, each one of STRIDE component violate one security attributes. CIA^3 (an extension of CIA with **A**uthorization and **A**uthenticity) attributes used as possible security requirements for each asset.

For each of the defined assets, a set of security requirements needs to be checked. These requirements are a reflection of the various threats which may threaten that asset. Thus, we check whether each asset suffers from one or more of the STRIDE model components. As we can see in Table 4.1, each of the STRIDE components violates one of the security attributes. CIA^3 (an extension of CIA with **A**uthorization and Authenticity) attributes are used as a source to determine the security requirements for each asset.

Integrity: providing integrity within vehicular systems is composed of:

- Providing *hardware* integrity to prevent and detect any hardware component fraud.
- Providing *framework* and *software* integrity to ensure that only trusted code can be run and to prevent infected code and malware from running.
- Providing *data* integrity to safeguard against any modifications to data during a transaction.

Authentication: is a special type of integrity which addresses the integrity of the data origin. In the case of transmitted data, authentication is about knowing who is communicating with whom. Authentication is essential to support authorization.

Authorization: determines whether a certain component is allowed to access or communicate a certain resource (i.e., who should talk to whom). Approving the request of that component depends on the authentication of the requester as well as the access control rules for the requested resource.

Confidentiality and Privacy: while providing authentication for the exchanged messages in the vehicular domain is vital, providing confidentiality is often less important. For example, there is no critical reason to encrypt all the messages exchanged between the different ECUs inside the vehicle. Enforcing confidentiality for the exchanged data should not be mainly to prevent vehicle identification detection. The ability to identify the vehicle is feasible already by different mechanisms without the need to snoop on exchanged messages (such as identifying the vehicle by color, number plate, etc.) The primary goals should be preventing a leak of the driver's critical data (such as driver behavior, previous location) as well as guaranteeing that any observer is unable to efficiently link different messages coming from the same source. In some scenarios, confidentiality is required; for example, leaving valuable stored information (e.g., private keys) without any confidentiality protection may leave the entire vehicle security at stake if an attacker is able to extract this data.

Availability: refers to the fact that the assets must be available even if the system is under attack. Availability is required particularly for safety-related applications which are integrated into the vehicle. Losing the availability of such applications may have serious consequences, and even threaten the lives of passengers. This property is a common concern of both safety and security, but they address it differently. While safety handles unintentional events which may lead to loss of availability, security focuses on intentional attacks.

4.2.5 Attack Accessibility

The way in which threat agents can exploit a vulnerability in any one of the aforementioned assets is very important for determining the applicability of the attack and proposing the right mitigation against it. We have specified the following three possible cases:

- **Direct access:** Some attacks require direct (physical) access by the threat agent to the target vehicle. Replacing the hardware component or connecting a malicious third party to the in-vehicle network is an example of such attacks. Such direct access could be achieved while a vehicle is parked. In such circumstances, the attackers cannot access the in-vehicle system, but may still be able (for example) to attach an external device to the vehicle such as a GPS device to track the vehicle later, or to target the vehicle's immobilizer and electronic locks [VGE15]. In some cases, taking the car to the service station for a regular check could become an avenue for direct access by attackers. In such cases, an attacker has full access to the in-vehicle system and could take advantage of existing physical interfaces (e.g., OBD-II port, USB port, and others) to gain direct access to the internal network. The owner or the driver has the advantage of log inside the vehicle for an unlimited time.

- **Remote access:** Other attacks do not require any direct access to the target vehicle. Attackers could target the vehicle remotely. Such attacks take advantage of the integrated wireless features of modern cars. These features include Bluetooth, a cellular connection, wireless tire pressure monitoring, etc. Attackers need to be within a particular distance of the targeted vehicle; this distance is based on the technology which used to attack the vehicle, e.g., 40 meters for wireless communication. Long-range wireless technologies give the attacker the ability to target the system from very far away, as in the case of using the entertainment system to play a song laced with malware which is able to emit malicious messages to the CAN bus [Kos+10].

- **Mixed access:** Direct access to the vehicle could be a means to introduce remote attacks. Indeed, some attackers with rapid direct access to the vehicle may install devices inside the vehicle (such as a USB cover, malicious DVD, malicious component connected via OBDII port, etc.) or outside it (communication sniffing devices). Later, they can employ those parasitic devices to target the vehicle remotely. Attackers may use other people to install these devices, such as a valet who parks the victim's car, a mechanic at a service station [Kos+10], and so on.

4.2.6 Abstract Model

In this subsection, we demonstrate how to apply the last step of adopting the SAVTA threat model. The outcome of this step is classification of as many known threats against vehicular systems as possible. Such a classification

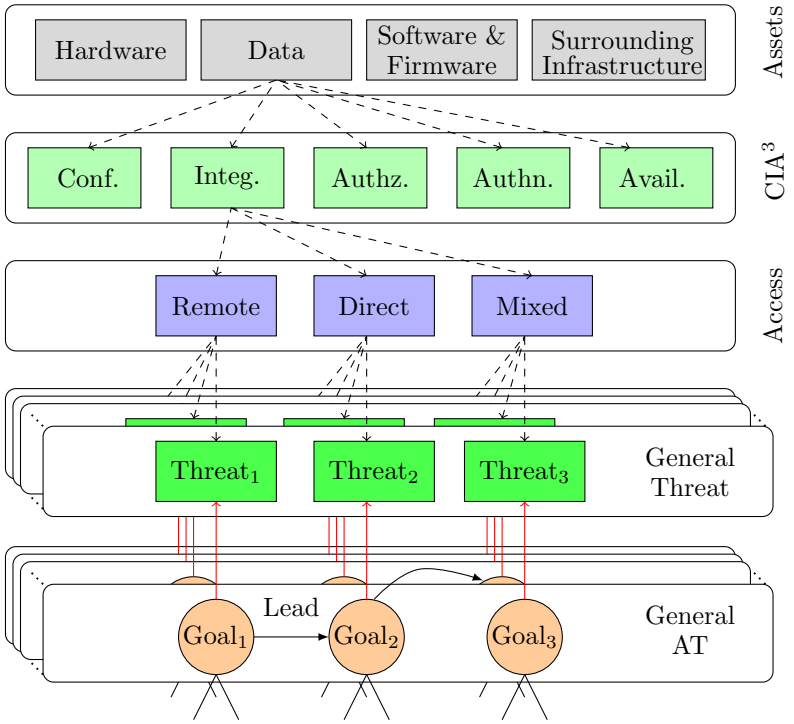


FIGURE 4.4: SAVTA method to identify and classify threats and links them to general attack trees

could reduce redundancy and inconsistency when applying defense techniques against homogeneous threats. In addition, it provides the basis for defining generic attack trees. Three layers were used to identify and classify threats (see Figure 4.4):

- Target assets: The first layer of the model contains all the (sub)system assets (e.g., hardware, software and firmware, data, or surrounding infrastructure). Within each of these assets, there are different vulnerabilities which could be targeted by motivated attackers.
- Requirements violation: The exploitation of an existing vulnerability in any asset will lead to a violation of one or more of the security requirements (i.e., confidentiality, integrity, availability, authorization, and authentication). We can further identify and classify potential threats based on the violated requirement(s).
- Accessibility: Eventually, the way of accessing the assets (i.e., remote, direct, or mixed access) in order to exploit a specific vulnerability is used as the last level for compartmentalization.

Applying this model to the entire vehicle system will identify all the known threats. For each asset a minimum list of 15 general threats needs to be checked. Each of these threats is used as the root of a general attack tree which explains how an attacker could exploit a defined vulnerability. For example, disabling one of vehicle sensors is a root for general attack trees.

TABLE 4.2: Factors for calculating the attack difficulty based on [Isoc; Isoa]

Factor	Description
Elapsed Time	The required time to identify a vulnerability in one asset and exploit it
Expertise	The level of attacker expertise (i.e. layman, proficient, expert, or collaboration of many experts)
Opportunity	the window of opportunity to perform the attack
Equipment and tools	The type of equipment and tools which is required to identify and exploit a vulnerability

Disabling such a sensor requires the existence of a specific vulnerability in that sensor which could prevent it from functioning if it is exploited.

Building the attack tree will help to illustrate attack vectors which threaten particular vehicle (sub)systems. These trees will turn into distinct ones, gradually reflecting the various studied subsystems. The accomplishment of one tree could open the door to fulfillment of other trees, as we explained within the stepping-stone attack. However, general attack trees seem to be indispensable for avoiding redundancy and interference between the high number of integrated sub-systems within the vehicle. The general trees will be derived from threats which were identified by our proposed threat model.

4.3 Risk Analysis

Attack trees have been used to evaluate the security risk to the system and calculate the difficulty and the probability of a successful attack. Determining the attack probability (i.e. likelihood) is related to the difficulty of identifying and exploiting attack scenarios. The great difficulty in performing an attack leads to a low probability of this attack occurring. This difficultly is dependent on many aspects (see Table 4.2) such as the time required for an attack, the desired attack tools, knowledge of system, and so forth [Isoc; Isoa; Hen+09]. However, regarding the risk analysis within the vehicular domain, calculation of the probability of potential attacks based on associating numeric values with each level of these factors may need to be reconsidered.

Elapsed time, for example, has a different effect in terms of the method of carrying out the attack (whether it is a remote attack or one involving direct access to the vehicle). Moreover, the overlap between expertise and tools employed also has different effects. Even inexpert attackers can launch an attack using sophisticated tools. Eventually, stepping-stone attacks should be considered during calculation of the probability of an attack. An attack might be unlikely, but achieving one attack goal in a different subsystem could increase its probability. For example, compromising the internal network of the vehicle could have a high difficulty level. However, the same attack become

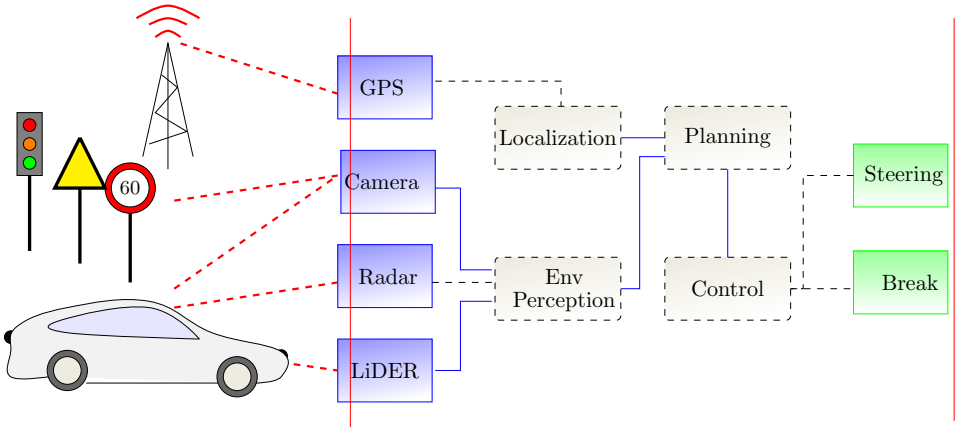


FIGURE 4.5: Hardware components and surrounding infrastructure in Automated Obstacle Avoidance use case

easier if we consider malicious devices attached by a valet while cleaning the car.

Furthermore, the motivation behind the attack should be considered when calculating the attack severity. The same attack could have varying results. Consider the following three attacks: (1) the driver re-flashes the ECU firmware (or replaces it with another type) to give the vehicle a more powerful performance. (2) A company which uses malicious firmware in one ECU to degrade the performance of a vehicle component or even lead it to produce misleading results intentionally (e.g., Volkswagen's emissions scandal [Gui+16]). And (3), a hacker or terrorist who manages to compromise the firmware of an ECU to steal the driver's information or to cause an accident. In all three cases, the attack is the same (i.e., compromising the ECU firmware) while the motivation and severity are different.

Finally, using risk assessment to identify the riskiest (sub)system and apply mitigation for that specific (sub)system is not enough to secure the vehicle. Many authors have argued that mitigating all the system vulnerabilities may involve significant costs, and therefore, particular vulnerabilities which have a low risk should be ignored, with the system accepting that risk. In any case, an attacker needs only one security vulnerability to break the entire system. Therefore, attackers will ignore highly protected (sub)systems and move to others which are not protected. Thus, the evaluation of the attack should play a role in determining the reaction to it but not whether or not we should mitigate it.

4.4 Use Case: Automated Obstacle Avoidance

We used our abstract model to identify the potential threats within the automated obstacle avoidance use case. Firstly, we need to define all components which could include vulnerabilities. We concentrate on the hardware components which are used within our use case. LiDAR, Camera, Radar, and GPS are possible (hardware) attack surfaces which can be used to target the

vehicle (see Figure 4.5). To keep the analysis simple, we decide to focus on one of these hardware components, which is the camera in this case. In the same time, we did not consider the software component which processes the captured image or the ECU where this component is mapped.

As we have described in section 4.2.6, after choosing an asset, we need to check the security requirements of the CIA^3 that could be violated by attackers directly or remotely when they target the selected asset. For the camera, we found that Integrity and Availability are the most targeted requirements. Attackers can disable the camera remotely (G_1) using different kinds of attacks such as a blinding attack ($G_{1.1}$) or jamming attack ($G_{1.2}$) as illustrated in [Pet+15]. The blinding attack can be performed by placing an LED device in a suitable place on the road ($G_{1.1.1}$) and using this device to emit intense light into the camera ($G_{1.1.2}$). This overexposed light prevents the camera from detecting the objects on the road. Performing such attacks requires an adequate level of skills and resources. The available evidences showed that security experts had performed such attack for research purposes. Also, disabling the camera can occur directly by covering the camera with dark tape ($G_{1.3}$) or even wrecking it ($G_{1.4}$). Confusing the functionality of the camera (G_2) could be another goal of the attacker, which could happen by damaging its auto control system ($G_{2.1}$). Such an attack does not require any sophisticated tools or skills. At the same time, it is easy to be detected.

Another case which could affect the integrity of the camera data is when the captured objects, such as road signs, traffic lights, and so forth, are mangled or removed. These components belong to the surrounding environment. We concentrate on the road signs component only. We determine the security requirements which could be violated as a result of attacks against these signs. The most common attacks which target road signs are removing ($G_{3.1}$) or distorting them ($G_{3.2}$) as in [Eyk+17]. Car thieves could perform such attacks. Some attackers could try to change the information on the signs ($G_{4.1}$) or even install new fake signs ($G_{4.2}$). Another exciting attack is the one in which an attacker used a drone equipped with an image projector to project fake road signs ($G_{4.3}$) as explained in [Nas+19].

Figure 4.6 shows the general attack tree for the camera component and the threat agents who may target it based on the abstract model and the analysis that we performed. Table 4.3 contains the meaning for each node of our attack tree. The figure also shows how the manipulation of the surrounding infrastructures has a direct effect on the functionality of different components in our use case. As a result of such attacks, the camera will deliver incorrect information to the perception module, which leads to improper planning and motions. The wrong planning process may cause the car to drive to unwanted places; for example, an area where the thief is waiting.

4.5 Summary

In this chapter, we created a comprehensive threat model based on the existing vehicle-related threat modeling efforts. The model tries to answer three main questions: (1) what are we facing? By defining threat agents who may

TABLE 4.3: Notations and meanings of nodes

Notion	Meaning	Notion	Meaning
G_1	Disabling the camera	G_2	Confusing proper function of the camera
$G_{1.1}$	Blinding the camera [Pet+15]	$G_{1.2}$	Jamming Attack [Pet+15]
$G_{1.1.1}$	Placing a LED device	$G_{1.1.2}$	Emitting a strong light to the camera
$G_{1.3}$	Covering the camera with tape	$G_{1.4}$	Breaking the camera
$G_{2.1}$	Confusing the auto controls of the camera	G_3	Disabling road signs
$G_{3.1}$	Removing the road sign totally	$G_{3.2}$	Distorting the road sign [Eyk+17]
G_4	Manipulating the road signs	$G_{4.1}$	Changing the data on the road sign
$G_{4.2}$	Installing a fake sign	$G_{4.3}$	Projecting images of fake road signs [Nas+19]

target the vehicle system and their motivation and tools. (2) What do we have? By classifying the different assets (e.g., hardware, software) and their possible threats. And (3) what do we need? By defining the security requirements of the defined assets.

Our model classifies and identifies the threats based on the targeted assets; the violated security requirements and the accessibility of the threats. General attack trees can be linked to each of the identified threats. We explored the automated obstacle avoidance use case while trying to classify the potential threats against it, based on our model.

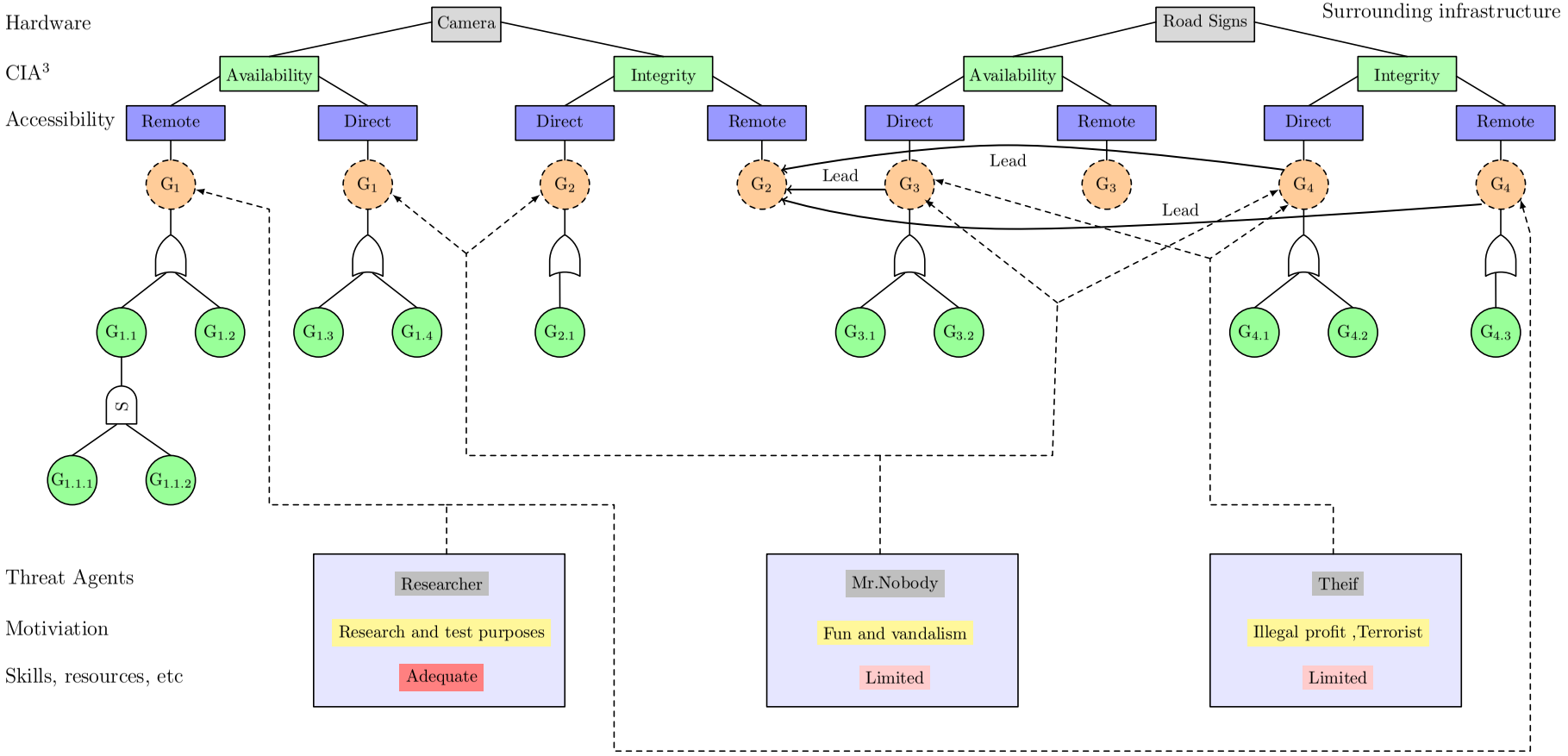


FIGURE 4.6: general attack tree for the hardware components in our use case

"Security is a process, not a product"
Bruce Schneier

5 Policy-Based Secure Communication

The heterogeneous nature of vehicle software components (varying in their levels of security auditing) and network architecture (many bus systems interconnecting with each other) have made the vehicle an attractive target for many attacks, as we have already discussed in Chapter 2. Communications within this heterogeneous and distributed environment which adhere to safety-critical and secure systems guidelines imply the formulation of a comprehensive and consistent communications policy.

Creating this policy is a complex, error-prone and labor-intensive task, requiring detailed knowledge of possible communication paths between all possible components within the system. For this reason, it is often skipped, trusting that each component will behave as intended and interact only with its peers. Traditional testing provides sufficient confidence to enable certification. Nevertheless, the existing process ignores malicious interference, whereby an adversary compromises a low-criticality process or subsystem and uses that to attack other subsystems, effectively taking over the vehicle.

In this chapter, we focus on fulfilling the security requirements (which were described in Chapter 4) to ensure the resilience of a vehicle system by securing the in-vehicle communication between the different software components mapped to various ECUs. To achieve this, (1) we propose a framework to build a secure communication policy. (2) We also propose a security module which acts as a connection policy checker, vetting incoming and outgoing communications and enforcing the distributed security policy.

The rest of the chapter is organized as follows. In Section 5.1, we present the challenges of developing a secure communication policy. The proposed policy framework and how it supports the software components updates are explained in Section 5.2. Section 5.3 details the proposed security module which is used to evaluate and enforce the developed security policy. The use

of the secure module to set up secure communication for in-vehicle communications is detailed in Section 5.4. In Section 5.5, we provide some measurements for evaluating our proposed secure module. Finally, we discuss our findings in Section 5.6.

Parts of this work have been published in [Ham+16; HP15; HNP17; PH15a; HP17]

5.1 Policy Development: Challenges

The correct definition of the security policy of each vehicular software component is a prerequisite for containing attacks against that component and preserving the security of the in-vehicle network. The security policy determines which actions can be performed by that component and which cannot. For example, it states which communications are authorized, how much of a resource can be consumed, which objects can be accessed, and so forth. In this chapter, we focus only on developing a security policy which rules the communication and access control for the different components. Defining such a policy is one of the main challenges for securing in-vehicle communications. It is a labor-intensive, challenging, and error-prone process, especially if we attempt to develop it during the final integration phases of a development process. It is challenging for multiple reasons:

- The high number of integrated ECUs in the vehicle and the complexity of the communication between the different applications on these ECUs. This may lead to configuration faults [Avi+04] due to setting of incorrect parameters which could enable unauthorized communication to happen.
- Late definition of security parameters for the different connections is very risky, because the chosen security parameters could violate the initial requirements of applications which were specified during the design phase.
- Moreover, vehicular system updates make the maintenance of the existing security policy labor-intensive and prone to errors. Each modification to the system (such as the addition of new functionality, moving a component from one ECU to another, or changing security parameters) requires updating the existing access control lists which can lead to reconfiguration faults [Avi+04]. Such faults may prevent authorized communication or allow unauthorized communications between software components.

5.2 Proposed Policy Development Framework

Our approach in addressing this challenge is to build the security policy gradually by integrating it throughout the design and life cycle of software

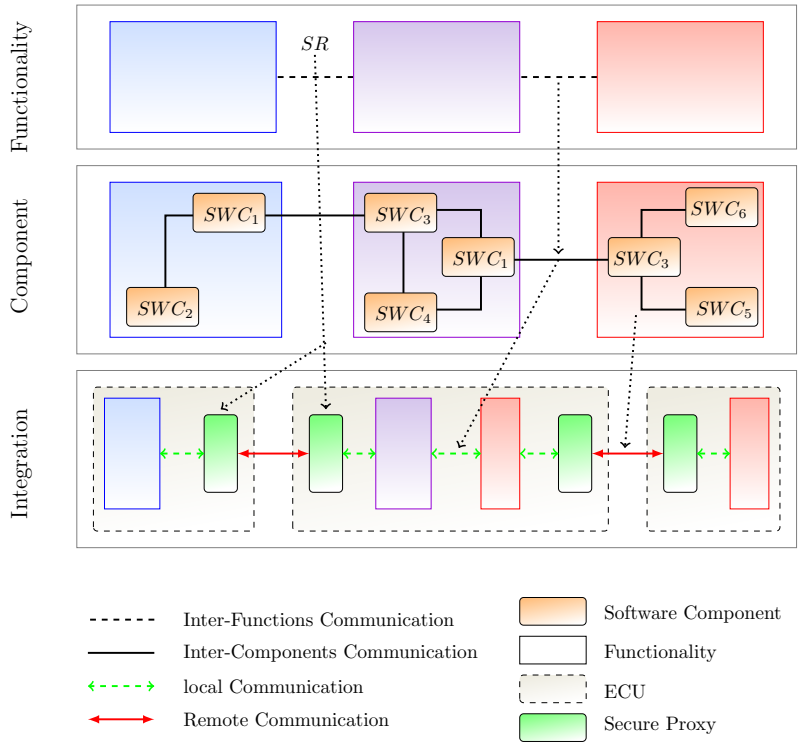


FIGURE 5.1: The three tiers of the proposed system architecture model to develop a secure communication policy

components. In doing so, the security policy will adapt to the changing circumstances during development, integration, and maintenance. Moreover, we will preserve the intentions of the initial designer and ensure the requirements of the current operational platform. We achieve this by ensuring the integrity of security requirements exchanged between the different parties during the various development phases. Policy-based communications allow the system to decouple the rules that govern an application's behavior from its functionality. This provides the flexibility to change the application-governed behavior without the need to re-code the system functionality after changes.

To achieve this, a system architecture model with three tiers is proposed. The tiers of this model reflect the life-cycle phases of software development based on the V-model and the SAE J3061 (Cybersecurity Guidebook for Cyber-Physical Automotive Systems). Our system model is based on similar frameworks which were used for some other goals, e.g., to decrease the complexity of automotive and embedded system software development [Bro06b; Bro+09; Fei+09; Thy+10], or to specify and maintain the timing constraints for different vehicle subsystems, as in [Sch11]. Within each tier there are different stakeholders and actors with different responsibilities, who interact and collaborate. Here are the three tiers:

- **Functional Tier**: Within this tier, all system functions are stated as black boxes within their defined boundaries. Here we have two boundaries; one represents the vehicle and the other represents every thing else.

The first step to define actual communication is by determining the interactions (or so-called *logical interconnections*) between these functions within the same vehicle, as well as their interaction with the outside environment (e.g., some functionalities may require data from the surrounding infrastructure or transmit some data to the outside world). Each interaction between any two functions can be defined as:

$$f_1(s) : s^{R_{sec}, R_{fun}}; s \in S_{f_2}; f_1, f_2 \in F \tag{5.1}$$

with s as an identical service required by the first function (f_1) and provided by the other (i.e., f_2). S_{f_2} represents a set of all the services provided by (f_2).

The main actor in this stage is the function designer (FD), who could represent the OEM. The FD is responsible for specifying the security requirements R_{sec} which need to be satisfied for each defined logical interconnection regardless of what the actual implementation of the link and the security mechanism might look like. R_{sec} may contain more than one security parameter, and in such a case, logical operators (e.g., &&, $\|$, etc.) are used to determine the combination of required parameters. For example, consider the case in which the FD states the need for *(I)ntegrity* and *(C)onfidentiality* for a certain logical interconnection. In such a case, we can express that as:

$$f_1(s) : s^{\{I\&\&C\}}; s \in S_{f_2} \tag{5.2}$$

In addition, FD can specify other requirements, such as functional requirements R_{fun}, for the defined link. One example of such requirements could be the maximum allowed time for the end-to-end latency.

Since we consider using a partially automated V-Model-like development process [Res+14], we assume that functional requirements are formulated at the beginning of the development process. The output of this tier should be a *functional system architecture* which determines all the logical interactions among the different functional blocks, including the services over which they interact as well as the various requirements for each interaction.

- **Component Tier**: This tier represents how each stated function is designed and developed. Every function in the vehicle is implemented through one or more interacting software components, as shown in Figure 5.1 (cf. [Sch+17]). Each software component encapsulates different software methods which provide its behavior and expose well-defined service interfaces.

  We assume that a software component can be instantiated multiple times to support different functions, for example library components for network access. (see SWC_3 and SWC_1 in Figure 5.1). The different components are implemented transparently from the actual ECU in which they will be integrated. In this level, the logical communication

which is derived from the functionality tier becomes more detailed. The services that the function requires and provides are linked to the exact software component which ensures or requires these services. Moreover, some implementation of security mitigation occurs in these tiers either as independent components which will be used later during the next tier or as part of the software component itself (e.g., validating the value of different input parameters for a certain software component).

The main actor in this tier is the software developer, which could be Tier 1 suppliers or a third-party company which is used by a Tier 1 supplier to outsource the development of the software components. Each Tier 1 supplier transfers the implementation of the functionality back to the OEM, along with all information related to the implemented components and the communication relations between them.

- **Integration Tier**: This tier represents the final stage of system development, during which developed function is mapped to the actual ECUs and the logical interaction translated to actual communication between two software components. One function can be distributed to multiple ECUs so that software components of the same function must communicate over the network (cf. Figure 5.1). In other cases, two functions could be mapped to the same ECU so that software components from different functions communicate internally.

  Eventually, when mapping the software architecture to a distributed system of ECUs, additional security requirements arise due to the need to communicate over the network. At this point in the design stage, security requirements will again be refined and annotated to the service interfaces. The same security requirement for two different logical interconnections can be ended with two different security methods based on the security proxy used (e.g., IPsec and capability passed). The OEM is responsible for performing the different integration phases.

These tiers ensure system composability, which provides the system integrator with several degrees of freedom for mapping software components from the processing chain to the available ECUs to satisfy different system requirements, e.g. by mapping a component to a remote ECU because of the restricted memory in the local ECU. Moreover, distributing the functional blocks over several ECUs due to the assumed transparent communication mechanisms requires an instantiation of network proxy components during the system integration process to enable transparent communication between network components.

To explain how this security framework could be applied, we revisit our use case of automated obstacle avoidance, which was explained in Section 3.2. The functional system architecture of the use case is displayed in Figure 5.2, where we can see the logical communication between the different functions. In order to generate trajectories for obstacle evasion, the vehicle must acquire and pre-process sensor data (three-dimensional point clouds in the case of LiDAR scanners). Pre-processing comprises filter and segmentation algorithms, ensuring that only relevant data is used to model the static

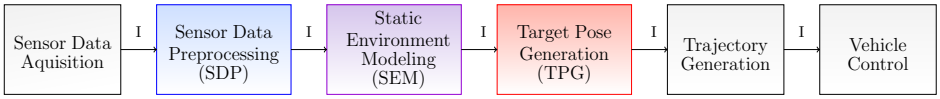


FIGURE 5.2: Exemplary functional system architecture of the automated obstetrical avoidance use case showing the different logical interconnection and the required security principle (i.e., (I)ntegrity)

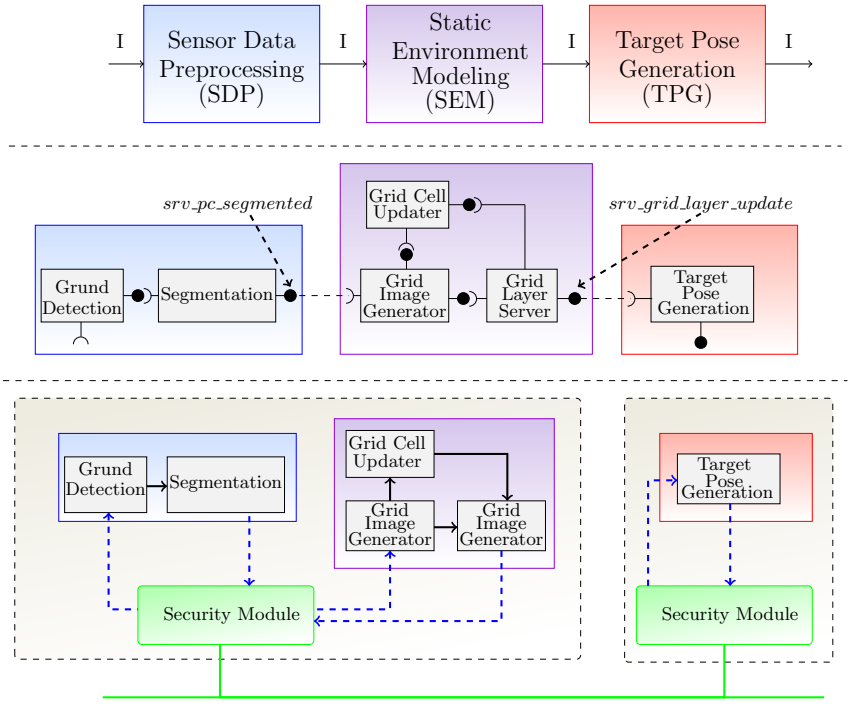


FIGURE 5.3: Software components of use case functions and platform mapping

environment. In the example application, the environment model consists of occupancy grid maps, which provide a map of the drivable and non-drivable areas around the vehicle. From these grid maps, a target pose can be generated for the vehicle. A trajectory towards this target pose is planned through the drivable areas of the environment. This trajectory provides reference values and set-points for the control algorithms which are responsible for maneuvering the vehicle safely through the static environment.

Figure 5.3 shows an extract from the functional processing chain of required software components, as well as a possible distribution of software components over several ECUs for the application at hand. During implementation, when software architecture is developed, the generic functional (or logical) interfaces are mapped to actual service interfaces which connect the different software components (cf. Figure 5.3). Required and provided services are specified and security requirements which arise from implementation-specific considerations can be attributed to the service interfaces.

LISTING 5.1: SEM services and required secuirity levels.

```
function SEM {
  services{
    provides srv_grid_layer_updated {
     security_level == Integrity;
    }
    requries srv_pc_segmented {
     security_level == Integrity;
    }
  }
}
```

LISTING 5.2: TPG required service.

```
function TPG {
  component target_pose_generator {
    services {
      requires srv_grid_layer_updated {
       security_level == Integrity;
       timing ==  2 ground_detection;
      }
    }
  }
}
```

In our example, the process for the development of the environment-perception processing chain would look as follows: The designer of the function for *Static Environment Modeling (SEM)* specifies that the function requires a service from *Sensor Data Preprocessing (SDP)* to receive segmented point clouds and provides a environment model over a service to *Target Pose Generation (TPG)* (cf. Listing 5.1).

In the same manner, the designer of Target Pose Generation (TPG) functionality states that the function requires a service ("srv_grid_layer_updated") (cf. Listing 5.2). It is important to note that at this stage the designer is not concerned about how the two ends of any connection will be mapped or the type of bus system over which the data will be exchanged. Moreover, the designer could determine the time constraints of the connection by specifying the maximum acceptable latencies (e.g., in this case maximum latency of 2 ms), as presented in [Hol+]. After having specified the functional dependencies, the actual implementation of the components can be annotated in the same way, providing information about which component provides and requires which services.

With the implementation under discussion, the software architecture can

LISTING 5.3: TPG communication policy after the final integration phase.

```
function TPG {
  component target_pose_generator {
    services {
      requires srv_grid_layer_updated {
        security_level == Integrity;
        security_protocol AH;
        bitrate y;
        ECUs {
          local ECU1;
          remote ECU2;
        }
        IPs {
          local ECU1_IP;
          remote ECU2_IP;
        }
        ports {
          local P1;
          remote P2;
        }
      }
    }
  }
}
```

then be mapped to different ECUs during the integration process. The integrator of a specific functionality takes the designer's policy for each component to check the required services for this component and search for the components that provide these services, which must be integrated. The integrator can access this information from a local repository which contains all the related-platform properties of the integrated components. The integration process may occur in multiple intermediate phases, during which more details could be added to the policy. At the end of these sub-phases, the logical and inter-component connections which were defined in the design phase are translated to actual communication mechanisms available on the platform. These include Inter Process Communication (IPC) mechanisms or network communication via a network proxy as shown in Figure 5.3.

In the given example, the integrator of the components implementing the TPG functionality updates the communication policy between TPG and GLS by adding the actual location (i.e., IP address and port numbers) of both components in the platforms. Moreover, knowledge of the components' mapping gives the integrator the ability to adjust the connection and specify the security protocol which is supported by the link (see Listing 5.3).

We note here that the local parameters (e.g., local IP, local port, etc.) refer

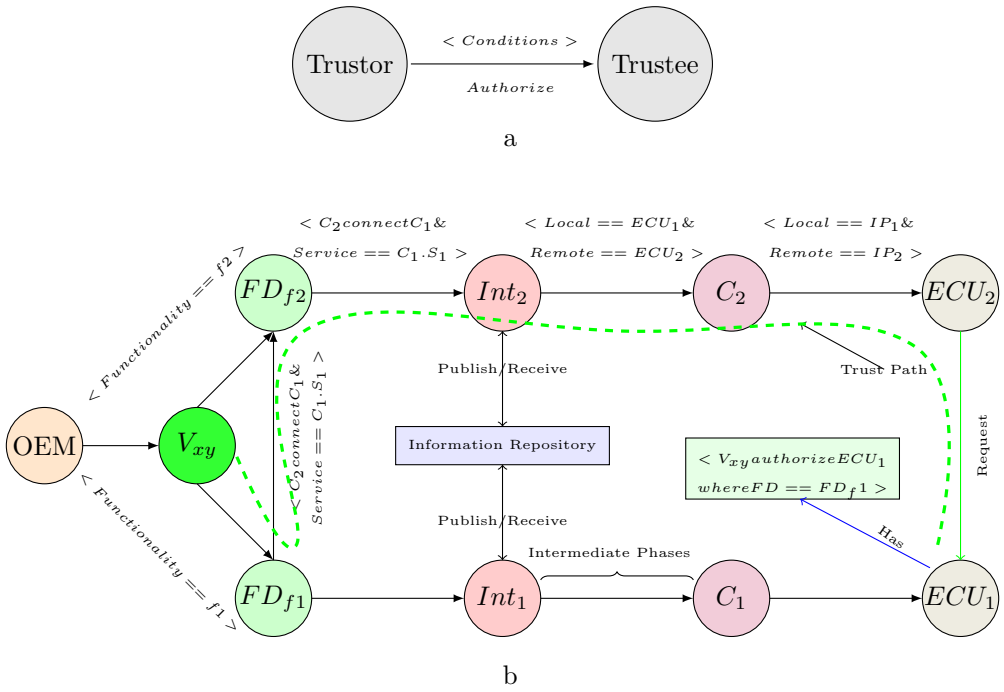


FIGURE 5.4: a: Trust relation between two entities, b : Trust management module between the different entities.

to the ECU where the request will later be evaluated (i.e., where the component was mapped).

5.2.1 Maintaining Policy Provenance

We have seen how the communication policy can be defined at the design level in an abstract way, and how policy parameters are modified incrementally during implementation. In such a hierarchy, the refined communication policy parameters need to be consistent with the existing ones from the previous phases. Therefore, ensuring the provenance of the communication policy during policy refinement is a crucial goal [Che11].

We construct a trust management module based on a Public Key Infrastructure (PKI) to build trust relations which provide delegation between the multiple entities who participate in developing the communication policy (e.g., a functionality designer (FD), Integrator (Int), etc).

Each one of these entities is identified by a public and a secret key. The trust relation is explained in Figure 5.4-a, where the trustor entity authorizes a trustee entity under specific conditions. The conditions could be a set of pairs separated by logical operations (i.e., &&, ||). Each pair represents a defined communication variable and its desired value, with operators such as $\leq$, $\geq$, <, >, $\neq$, or == used to express the relationship between each variable and its value. The security credential could be derived from the trust relation as in Listing 5.4.

The security credential, which states the public keys of the trustor and trustee, as well as the conditions, is signed by the secret key of the trustor entity to ensure the policy integrity. The goal of creating such a policy is

LISTING 5.4: Security credential derived from trust relation.

```
Trustor: Trustor_PK
Trustee: Trustee_PK
Condition:{ var_1 == val_1 && var_2 > val_2}
Signature: Signed with Trustor_SK
```

different than the purpose of a certificate [Hou+02]. A certificate only authenticates the owner's key; it does not prove its trustworthiness.

Figure 5.4-b illustrates trust and delegation between the different entities. For example, in the vehicular domain, an OEM starts the production of a specific vehicle model (e.g., V_{xy}), requiring a set of functionalities. Each designer of these functionalities is authorized to provide its functionality. Whenever a connection is required between components in different functionalities, a chain of trust is established between the two functionality designers. Later, the trust is passed to the integrators, applications, and the platform where the applications run. By building this trust we create a chain of trust between the requester and trusted root which is defined in the local policy of the receiver. The receiver authorizes the request only if a trust path between the requester and its trusted root can be found.

5.2.2 Updating a Component Scenario

In the previous section, we showed how the creation of the security policy could take place in the lab during the production of the vehicle. However, in-field updates are becoming ever more popular, as demonstrated by the recent over-the-air updates Tesla cars and others [Res+14]. Regarding security, the platform should thus ensure the integration of new components without putting the vehicle at a risk of violating the system requirements or creating a vulnerability regarding communication and access control rules. We propose an integration procedure, as shown in Figure 5.5.

The platform receives a application description package which contains the new software component, a signed hash value of the software component which ensures the authenticity and the code integrity of the application, the design-level credentials, and integration specifications which specify where the component should be mapped. The system's middleware verifies the received software component then checks the required services that the new component requires (based on the design-level credentials) and determines whether the required application is mapped to the same platform or to a remote platform. The integrator component provides an identity for the new application and publishes its information (public key, service name, hosted ECU, etc.) via a service name repository. Each ECU contains a service name repository to provide the information about the other services and to broadcast the information of the newly added component to keep the other repositories updated. Regarding the platform where the remote application was

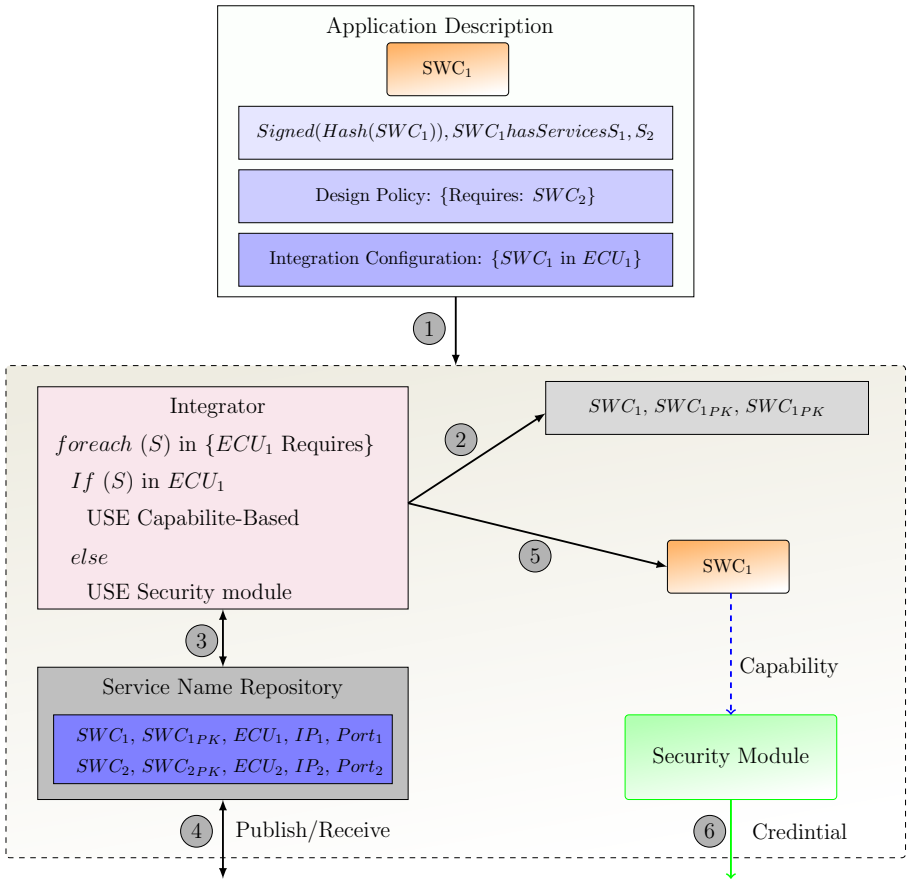


FIGURE 5.5: Communication policy during updating/installing an existing/a new component.

mapped, the integration component provides the application with the capability to communicate with local applications or with a proxy with required credentials to use the network as we will explain in the next section.

5.3 Distributed Security Module

The distributed nature of a vehicular E/E system's components makes the use of single ECU for evaluating the communications credentials and enforcing the communication policy infeasible. Therefore, we aim to enforce communication policy in a distributed manner by adopting a distributed firewall technique (cf. [Ioa+00]) so that we equip each ECU with its own security module which acts as a connection ingress policy checker by vetting incoming and outgoing communication, and enforces the distributed security policy locally.

In this section we explain how we develop such a distributed security module and the different components it includes.

5.3.1 From IPC towards networked communication

Any microkernel architecture provides strong isolation of application components in order to minimize the trusted computing base. Therefore, any communication between isolated components (i.e., IPC) needs to be mediated by the kernel. On the one hand, this introduces an additional overhead, which was historically one of the main drawbacks of the microkernel approach, but was weakened by the optimization of (synchronous) IPC mechanisms and the evolution of microkernels [Lie93; EH13]. On the other hand, this has the benefit of making any communication explicit. This property was further strengthened by introducing capability-based access control that enables fine-grained and unforgeable control of a component's communication channels. As a result, today's microkernel architectures allow us to apply the principle of least privilege and to enforce security policies when integrating application components from different, potentially distrusted parties [LW09]. In summary, all these properties helped establish microkernels as a sophisticated, and also commercialized [Gen19; Okl; Qnx], implementation for critical application domains such as vehicles.

When it comes to more dynamic scenarios like distributed systems, a service-oriented approach is commonly taken to equip the system with the required flexibility. Typically, a communication middleware then takes care of routing the messages to the communication partner, which registered under a certain service name, thereby deploying a communication mechanism (API) that is agnostic regarding the actual communication partners and their location. However, a major drawback of such middleware is that enforcing security policies and providing isolation (e.g., local namespaces) without adding a significant overhead is a non-trivial obligation. This approach clearly trades ease-of-use against simplicity and efficiency.

We therefore believe that the strong architectural guarantees already provided by microkernels can and should be utilized in such scenarios. That means local communication shall still benefit from the existing efficient and secure implementations while we transparently transform the local and remote communication mechanisms where necessary. The challenge here is to provide similar guarantees for remote communication, i.e., the fine-grained access control and integrity of remote communication channels. Yet there is a mismatch between the fine-grained access control for local IPC and the socket API typically used for network applications which give full access to any attached network. We therefore need to provide the infrastructure with which we can control network accesses in order to establish unforgeable network communication between application components, as we are used to doing when using local IPC.

5.3.2 From user-level networking towards a distributed firewall

Microkernel philosophy is based on moving all components, including device drivers and protocol stacks, from the kernel to the userspace. Therefore,

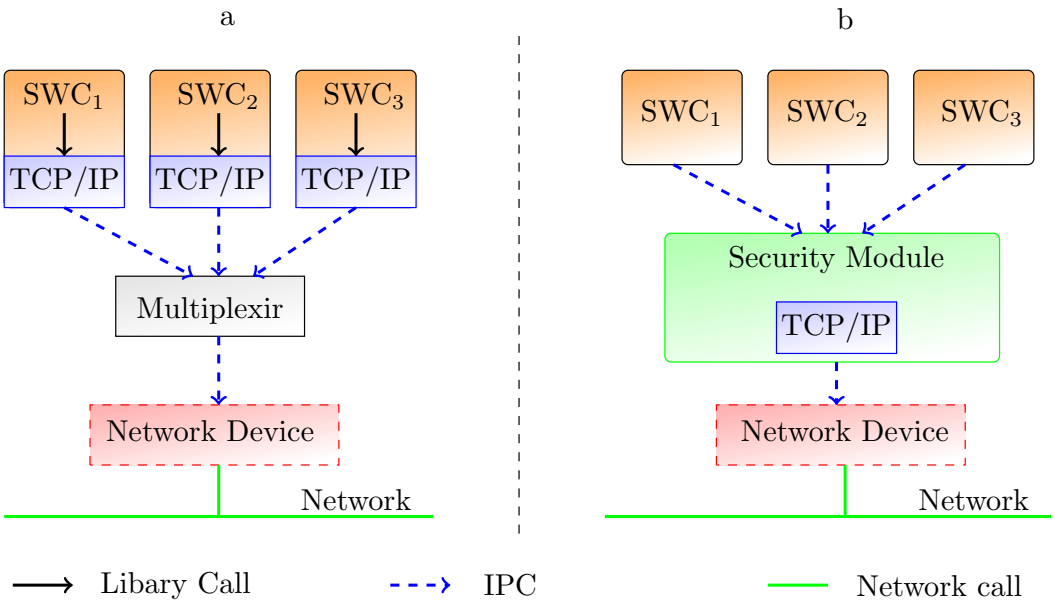


FIGURE 5.6: Architecture with multiple TCP/IP stacks and a shared multiplexer (a) compared to a security module with a shared and integrated TCP/IP stack (b).

implementing the network stack in the userspace was a hot topic for many years; it was proposed for different motivations, including increasing the performance and flexibility of the network layer [Eic+95].

Providing maximum isolation between different applications, a straightforward approach of executing several network applications on a microkernel consists in using dedicated network stacks for each application (cf. Figure 5.6-a). Here, the low-level Network Interface Controller (NIC) must be multiplexed/virtualized, e.g., by a network bridge. As a result each application is linked to its network-stack library and requires its own MAC and IP addresses. The drawback of this approach is that unless some sort of packet filtering is deployed, each application also gains full access to the shared network, which contradicts our objective of fine-grained access control. More precisely, this approach is susceptible to the following communication threats:

- **Spoofing:** An application which has full access to the network stack, can emit a frame with fake IP or MAC addresses. Such an application may imitate other applications, eavesdrop on their communication, or collect relevant information about the platform. It could also change the transmitted data and inject false values.

- **Denial of service (DoS):** One of the primary results of IP spoofing can be a DoS attack, i.e., a malicious application could spoof a target service's IP address and send many packets to different receivers. All responses to the spoofed packets will be directed to the service's IP, which will be flooded. Sometimes an attack cannot cause disruption to the service, but can cause a degradation of its quality (e.g., by increasing its

response time). The DoS could lead to serious issues if that service is responsible for users' safety.

In order to combat these threats, adequate access-control mechanisms should be implemented to control the interaction between different applications and to prevent unauthorized parties from processing foreign data. However, packet filtering is more network-centric and typically too abstract for fine-grained application-level access control. Moreover, there is a consistency challenge when it comes to updating static filtering rules in a distributed system in the context of a dynamically changing environment.

Hence, adopting the distributed firewall technique [Ioa+00] seems to be a favorable solution to remove any performance bottlenecks. We applied this method by providing a single security module for each ECU, as shown in Figure 5.6-b. This module is playing the role of a firewall by controlling all incoming and outgoing communications on a single ECU and by enforcing the security policy locally.

As a system integrator, we can thus perfectly control the local inter-component communication and thus guarantee that no application component has direct access to the network interface. Moreover, the security module is able to distinguish its clients by their capabilities and can therefore select and enforce different (pre-defined) policies for network communication. In this way, all network accesses are securely mediated by the security module. Note that this is based on the assumption that capabilities cannot be arbitrarily delegated between application components.

The security policy is managed centrally and then distributed to all ECUs. Note that the security module implements a shared network stack and multiplexes the network device. It is therefore a potentially complex component that might compromise the isolation of the application components. We believe, however, that this design choice can actually simplify the policy enforcement and multiplexing task in contrast to solutions that implement these on other layers of abstraction.

5.3.3 Security Module Components

The security module is composed of four cooperating components as depicted in Figure 5.7. In the remainder of this section, we elaborate on the design and implementation of these components in more detail:

Proxy

This component plays a central role by providing a suitable interface to the applications as well as by coordinating the other components. Many conventional (legacy) applications use a socket-like API (as in the standard C library) to access the network stack. The proxy represents the interface which the applications use to interact with the security module. In contrast to the conventional function/library calls, the proxy uses local IPC and must therefore take care of the memory management between the different address spaces

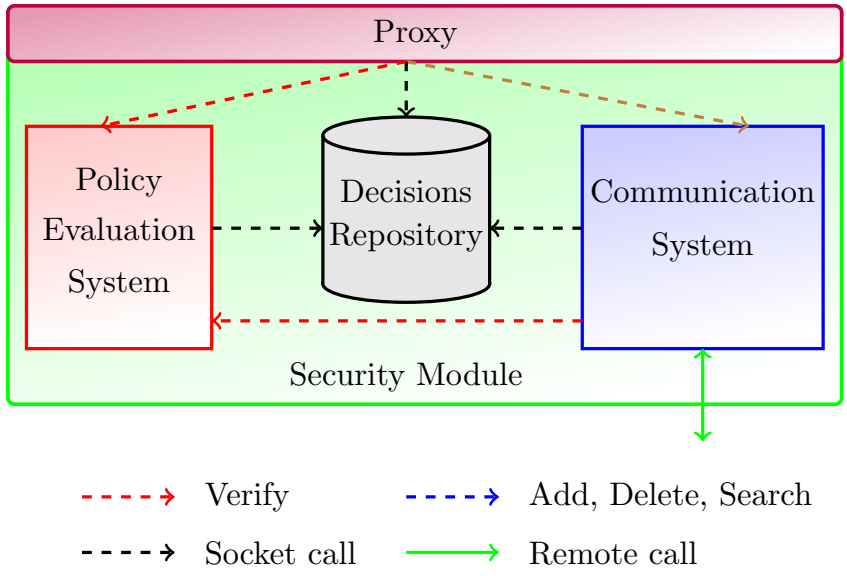


FIGURE 5.7: Architecture of the security module

of the communication system and its clients before a call can be handed over to the actual network stack.

More precisely, shared memory was used to transfer the data between the address space of the application and the security module to avoid imposing extra overhead by copying data multiple times. All this is transparently taken care of by this component, so that the clients can still use the typical socket API functions. Legacy applications can be supported by linking against a slightly modified version of the standard C library that forwards the socket API calls to the security module.

Policy Evaluation System (PES)

This system is responsible for monitoring (i.e., granting or denying) an applications' main requests, such as initiating a connection, and sending or receiving data. In order to make a decision, it determines whether a proposed request is consistent with the global system policy, which represents the least accepted security parameters and system-wide security configuration, and whether the conditions specified in the communication policy were met. This system depends on the KeyNote trust management engine which is used to evaluate the potential action of each application. Each application has a communication policy (we can also refer to it as a credential) which gives it the ability to communicate with other components, regardless its topological location in the network. The communication policy of each component, in principle, should not conflict with the global policy.

The KeyNote policy definition language [Bla+99] is used to formulate the communication policy. The application-independent design of KeyNote allows for the support of a variety of different applications. KeyNote furthermore enables the delegation of the policy by allowing principals to delegate

authorization to other principals. This fits perfectly with the policy development framework which was presented in Section 5.2. Consequently, the delegation capability enables decentralization of policy administration.

Communication System (CS)

As mentioned before, the network stack was integrated into the security module to provide basic network access. In our module, we have used the LwIP for this purpose. In addition, we integrated embedded IPsec [SK05] into this network stack (as proposed in [HP15]) in order to provide basic security services (e.g. integrity, confidentiality) to the various software components.

IPsec may be implemented in a host using several methods. It can be implemented through the bump-in-the-stack (BITS) method by implementing IPsec as a separate architectural layer between the IP and the data link layer. Another method of implementation which we adopted, is integrating IPsec into the IP layer [Koz05, p. 455]. In our implementation we used an existing package developed by Schild et al. [NC03]. They adopted the BITS method for their implementation. This package has a number of deficiencies: it supported the tunnel mode only while the transport mode was not implemented. Moreover, it supports only manual keying to set up the Security Association (SA) parameters. Finally, it does not handle the problem of IPsec with fragmentation; the package can handle packets whose size is less than the maximum transmission unit (MTU). The previous implementation drops the packet silently. In the case in which the protected packet exceeds the MTU, this action may later stop the entire communications.

We enhanced their package by implementing the transport mode for both AH and ESP. We also removed the implementation of MD5, SHA1, and 3DES-CBC algorithms from the package since they are already part of OpenSSL library which is ported to the Genode framework. Adding fragmentation to the IPsec implementation could be achieved in two ways: (a) by continuing to use the BITS architecture and re-implementing the fragmentation services to keep the IPsec package independent or (b) by integrating the IPsec into the IP layer and using the existing fragmentation code. We selected the second option in our implementation.

Decisions Repository (DR)

This component provides a repository for saving policy decisions. Such a repository is an essential technique in our design to spare the run-time costs of request evaluation. By doing this, evaluation only occurs when an application initiates a connection (i.e., uses accept and connect calls for TCP-based communication). For this purpose, the decision repository stores the decided rules for any opened connections. These rules contain all the information that the IPsec protocol needs to process ongoing and outgoing packets.

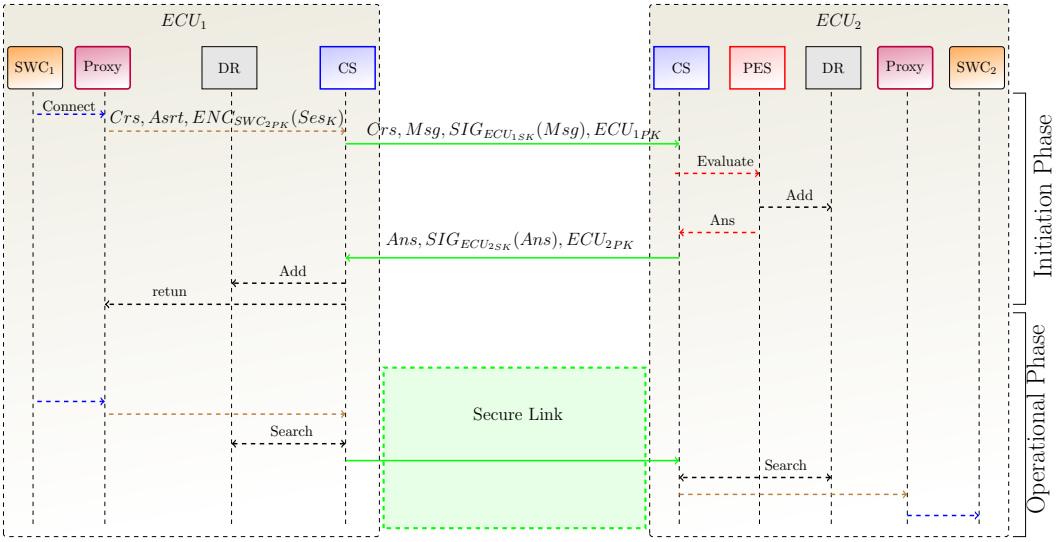


FIGURE 5.8: Secure connection setup.

5.4 Secure Communication

The security module within each ECU ensures secure communication links for all software components mapped to it. Before we reach that point, a connection initiation needs to take place; Figure 5.8 illustrates this process. To explain this, we consider the case where SWC_1, which is mapped in ECU_1, wants to communicate securely with SWC_2, which is assigned to a different ECU (i.e., ECU_2).

5.4.1 Initiation Phase

During the initiation phase, whenever SWC_1 on ECU_1 establishes a connection (e.g., it uses a connect socket call), the *proxy* intercepts this call and redirects it to the *CS*. The *proxy* also delivers all required credentials (*Crs*) to vouch for the local ECU to communicate with the remote ECU. It also prepares all the required assertions (*Asrt*), which will be needed by the remote counterpart to evaluate the request. A session key (Ses_K) could be created and delivered to the proxy. This key will be used later as a symmetric key to secure communication between SWC_1 and SWC_2 after validating the request and when the initiation phase is over. This key will be changed at a later point at regular intervals. This key is encrypted (*ENC*) by the remote application public key (SWC_{2PK}) to prevent any third party from perceiving it during the request transmission.

The *CS* on ECU_1 uses the private key of ECU_1 (i.e., ECU_{1SK}) to sign (*SIG*) the transmitted message (*Msg*) to the remote ECU_2 to ensure communication integrity and message origin authentication. The *Msg* contains the received assertions (*Asrt*), and the encrypted session key, alongside a monotonic counter value. This value is used to guarantee that messages are up-to-date and to prevent replay attacks. A dedicated manager usually generates this counter value as in [AUT17]. The mechanism of creating such a value is out of the scope of the thesis.

The signed message, the ECU_1's public key (ECU_{1PK}) and the credentials are all sent to the remote communication counterpart (i.e., ECU_2). Note here that we sent the credentials without signing them because they were already signed and require no further protection. The security module in the ECU_2 delivers the received request and the credentials to the PES to evaluate them based on the local communication policy. If ECU_2's PES authorizes the request, a new security rule will be added to the DR.

Each inserted rule includes all the communication identifiers such as the local and remote IPs, local and remote applications ports, remote public keys, the required security service (i.e., integrity, confidentiality, no security), access control role (i.e., authorize, deny), an identifier for the session key, and others.

The location where the Crypto keys (i.e., SWC_{2SK}, ECU_{2SK}, Ses_K, and other secrete keys) is stored and the different crypto operations are performed is a very critical aspect. We propose to avoid saving the keys directly in the DR. Instead, an identifier of each key should be used to refer to the actual key. We recommend the use of a dedicated crypto module, which will be responsible for all crypto operations and for storing the keys. Or placing the crypto code in a trusted environment, e.g., ARM TrustZone technology.

Later, CM on ECU_2 sends the answer of the request to the remote CM. The answer is signed by the ECU_2's secret key (ECU_{2SK}) to prevent any modification of the response by third parties and to give ECU_1 the ability to authenticate its origin. In case of authorized connections, one rule is added to DR on the ECU_1.

5.4.2 Operational Phase

After the initiation phase is done, both software components can exchange data securely. The secure connection here means that IPsec protocol is involved to ensure data integrity (by using AH protocol), or to ensure the confidentiality of exchanged data (by using ESP protocol) or even both principles. Also, secure connection implies that SWC_2 receives messages only from authorized software components (SWC_1 in our example) while the CS will ignore other messages. The CS represents the first policy enforcement point. It intercepts each incoming and outgoing packet selectors (i.e., IP, Port, etc.) and applies the relevant security rules (i.e., pass or deny).

It is important to note that the role of PES is only limited to the connection initiation phase to check the validity of the request. Whenever this phase is finished, PES does not interfere.

The second enforcement point in our security module is the *proxy* which provides an efficient and flexible access control layer to enforce the security policy between the applications on the same platform. The use of Genode allows us to leverage its efficient message-passing capabilities for the enforcement of the socket APIs of the different application. We can achieve that by replacing direct calls with IPC that are implemented via the Genode message-passing mechanism. The implemented proxy is responsible for

TABLE 5.1: The code size of the secure module's different components

Component	SLOC
Proxy	500
Policy Evaluation Module Interface	300
IPsec Extension of the Network Stack	2000
Decisions Repository	600

translating the socket calls to the proper IPC call and redirect it either to the *CS* or to a local application in the same platform.

5.5 Evaluation

In this section, we present the results of a number of experiments on prototype implementation to validate our proposed secure module. The goals of the experiments are:

- Determining the introduced overhead of the security module in terms of source lines of code as well as the effect of using a single network stack (as in our security module) instead of multiple ones (as in Genode framework by using nic_bridge components).
- Measuring the overhead of setting up secure communication as explained in Section 5.4.1.
- And finally, measuring the overhead of the embedded IPsec which we implemented to ensure the integrity and confidentiality of communication between different ECUs.

The rest of this section details the result of each of the abovementioned points:

5.5.1 Network System of Security Module

We evaluated our implementation of the security module with regard to its overhead in terms of source lines of code (SLOC) of the different components. We also measure the latency of the network system component of our module without considering any interference with other module's components.

Table 5.1 summarizes the SLOC values that have been acquired by the CLOC tool [Dan09]. Note that we only evaluated the part of the policy evaluation system which interfaces the unmodified KeyNote library. It is also worth mentioning that our approach saves about 750 SLOC by superseding the multiplexing component (i.e., nic_bridge).

Regarding the latency, we used the Netperf benchmark [Jon+96] in order to measure the network performance of our secure module. Netperf works

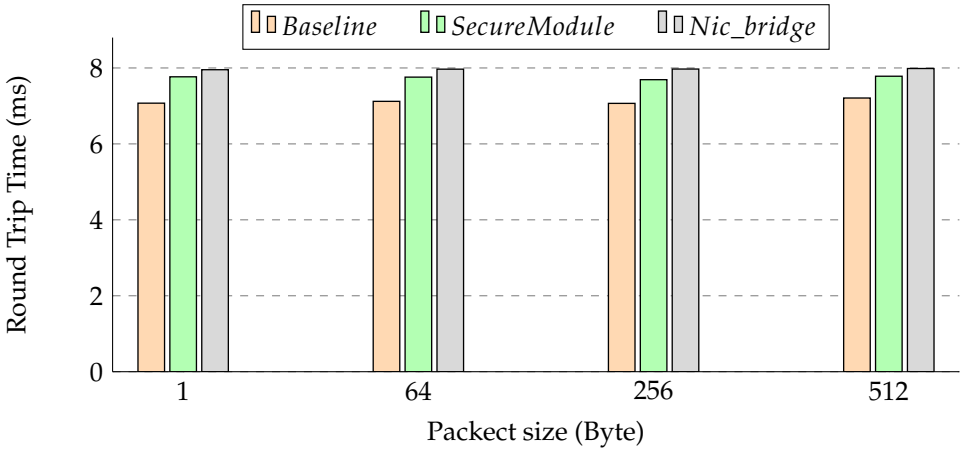


FIGURE 5.9: Average round-trip latency results for our production platform

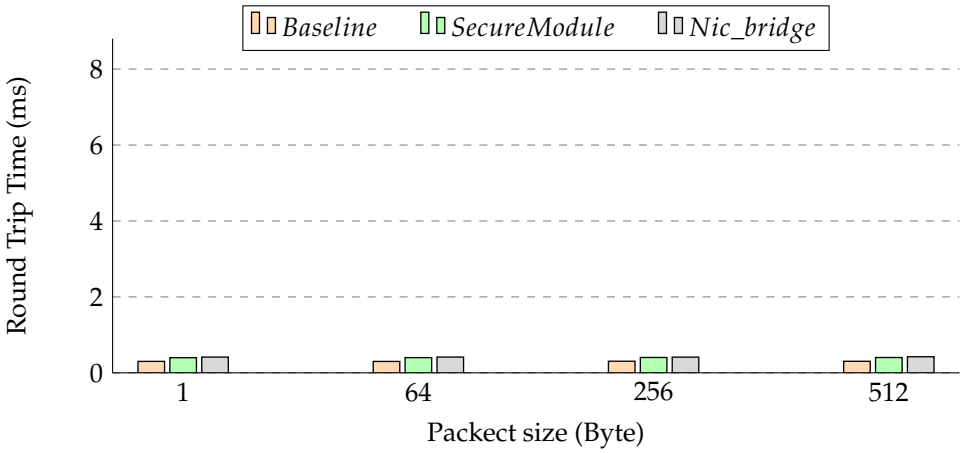


FIGURE 5.10: Average round-trip latency results for our security module running over Linux OS

based on the client/server model. It consists of two applications; the first one is the *netserver*, which runs in the local host and waits for the connection from the remote host. The remote side runs the second application, called *netperf*, which is used to transfer the test configuration to the *netserver* application and receive the result from it.

As a matter of course, the network system within our security module must perform worse than a scenario where a single application directly accesses the network device. Note that at this stage, we do not consider the interference of IPsec protocol. We therefore compared our approach with the scenario illustrated in Figure 5.6-a. The Genode OS framework proposed the use of a nic_bridge which implements the Proxy-ARP protocol [CMQ87] to multiplex and monitor the communications of different applications that run on the same host. Neither solutions use filtering mechanisms which identify the application properly.

Hence the scenarios we compared both include additional copying and context switching caused by the nic_bridge and the security module. The

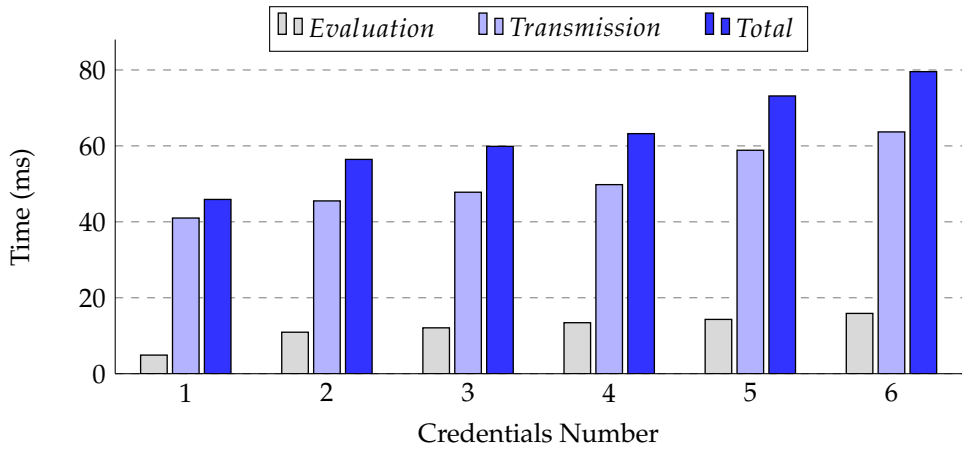


FIGURE 5.11: Performance evaluation of the security module.

netserver component was executed on the system under test while the *netperf* binary was run from a standard linux machine. In particular, we used the TCP_RR with the parameters `-i 10,3 -I 99,5` in order to perform multiple iterations and achieve a confidence level of 99 %.

The average round-trip latency results are shown for different package sizes and two different platforms in Figure 5.9 and Figure 5.10. As a baseline, we also included the results for a setup in which the *netserver* directly accesses the network device. One of the test platforms (production platform) was running Genode on a Raspberry Pi as described in Section 5.4.1 whereas the other platform was running Genode directly on the same Linux machine as the *netperf* binary. Note that we used the latter to bypass any physical network devices and drivers as it only utilizes rather simple virtual network interfaces.

From Figure 5.9, we can observe a slight improvement in the latency for our approach in comparison to the case where each software component has its network stack which shares the network device with other components via nic_bridge as it is adopted by Genode framework.

5.5.2 Initiation the Secure Connection

The next step was to evaluate the initiation phase of secure communication between the two applications run over RPIs using our security module. Both platforms are running Genode OS and our framework and communicating over a local network. The overhead is introduced from two parts: the necessary time to transmit the required credentials and time overhead of the request evaluation. At the same time, we want to show the effect of the number of credentials to evaluate transmission and request evaluation times. In this respect, Figure 5.11 shows the overhead observed from transmitting and evaluating the credentials. Each time we used a different number of credentials (i.e., from 1 to 6) to evaluate the performance.

We were able to notice that increasing the number of delegations (i.e., credentials numbers) introduces more overhead in both parts (i.e., evaluation

TABLE 5.2: The size of required files during the initiation phase in KB

component	size KB
1 credential	1.2
Public Key	0.38
Private Key	1.6
Policy	0.5
Signed request	0.8

and transmission). Most of the evaluation overhead comes from the signature verification of the credentials; we can lessen this overhead by caching the verified credentials and used it later without the need to verify its signature again.

Other than that, we evaluated the overhead regarding the size of the transmitted messages. Table 5.2 defines the size for each required file to set up the connections (i.e., credentials, public key, signed request). From the same table, we can estimate the required saving size for each application within 1-level of delegation which is around 3.7 KB (i.e., the size of credential, keys). we can estimate the size of the transited data for the evaluations which is also around 2.9 KB (i.e., credential, public key, signed request).

5.5.3 Using the Secure Communication

The last experiment was to measure the overhead of using our security module to secure the network while using IPsec protocol. The IPsec configurations for the measurement were as follow:

- AH protocol: we configured the test bed to use MD5 hash function combined with HMAC as a keyed authentication mechanism.
- ESP protocol: we set up the testbed to use both MD5 as an authentication algorithm and 3DES-CBC as encryption algorithm.
- IPsec for both AH and ESP was configured to use transport mode.

These configurations were hardwired in the IPsec code. The secret key which is used by IPsec is the session key (Ses_K) which was exchanged during the initiation phase.

Figure 5.12 shows the network latency of our security module. The latency was measured as a function of the size of the different transmitted packets while applying AH, ESP, and without IPsec (we refer to it in the figures as NoSec). Figure 5.12 shows that applying ESP protocol introduces more overhead compared to the use of AH protocol. However, the difference in the latency values between the case of using LwIP without IPsec and using IPsec (whether AH or ESP) is still negligible in most cases (less than 0.06 ms).

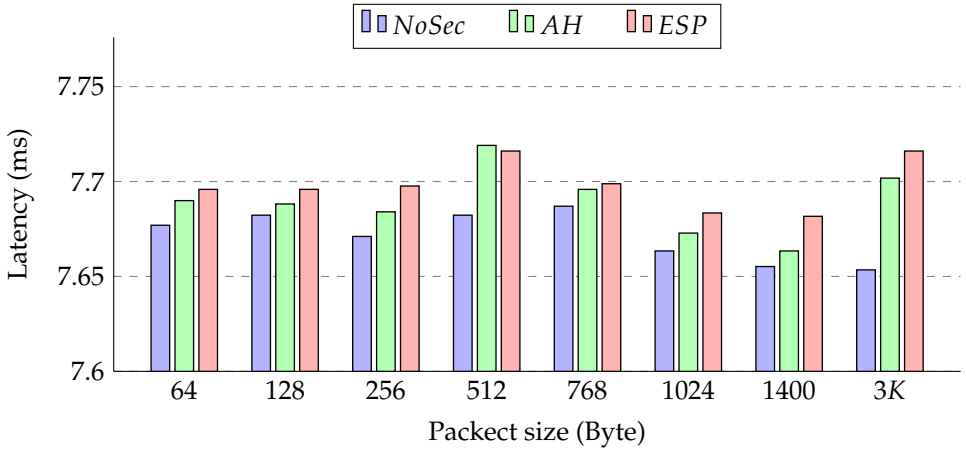


FIGURE 5.12: Latency of our security module while using AH, ESP, and no security protocol

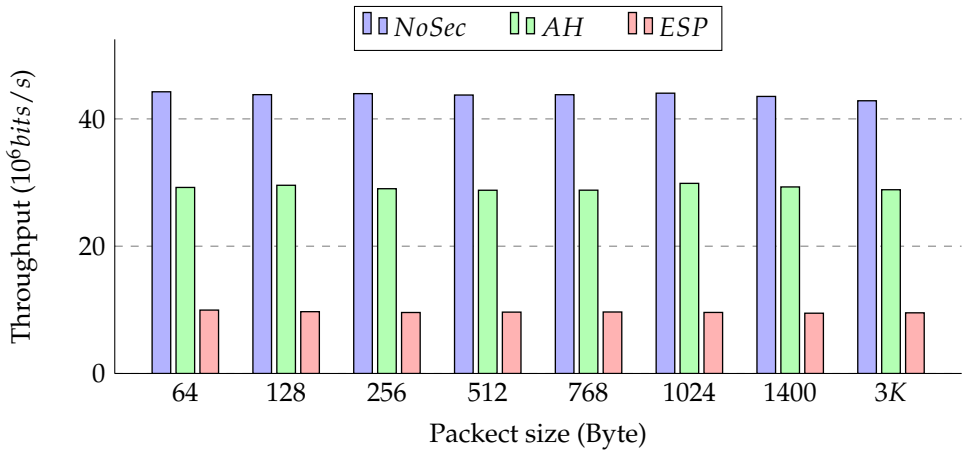


FIGURE 5.13: Throughput of our security module while using AH, ESP, and no security protocol

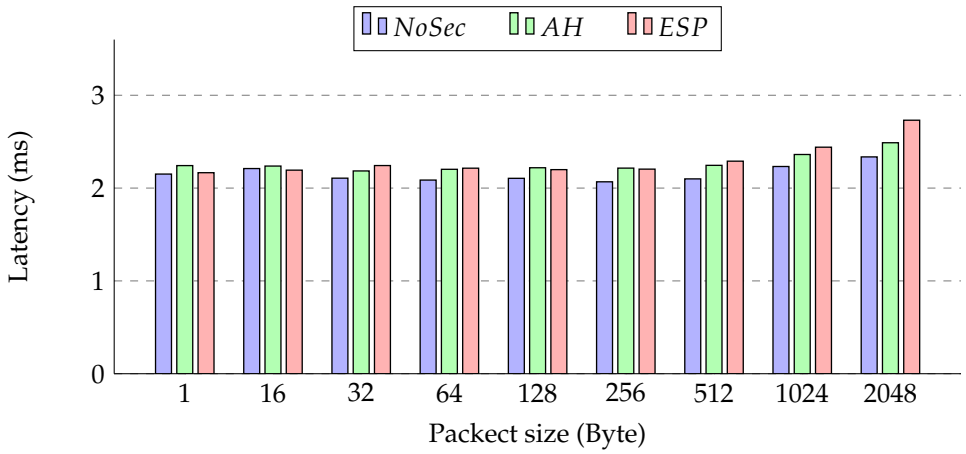


FIGURE 5.14: The Round Trip Time using Linux

In addition to latency, we were interested in measuring network throughput. Throughput is defined as the amount of data that can be transferred from the source to the destination in a specific period. Figure 5.13 represents the result of testing throughput as a function of the size of the different transmitted packets in the same previous three cases (i.e., IPsec with AH, IPsec with ESP, and NoSec). Based on the Figure 5.13, we can indicate that the throughput decreases 33% when we use AH compared to its value without using IPsec, while it decreases 77% when we use ESP compared to the throughput value without using IPsec.

It is important to note that our IPsec implementation did not cause the low network performance of the security module. When we measured the throughput of the system without our IPsec code, we obtained close results when comparing to our results in the case of using NoSec with our implementation. The next step was to check the effect of the hardware platform. Therefore, we ran the same measurement, but instead of using RPI, we run Genode (including our security module) directly on the top of Linux machine. Figure 5.14 shows the results of this measurement. As we expected, the network latency is far less compared to the case when we used RPI. Also, the introduced overhead of using IPsec was relatively similar to the case using RPI.

To discover the reason for low throughput values during the use of security module over RPI, we checked to see whether the network is facing the packet loss issue; we found that the rate of packet re-transmission was negligible (2%), so it cannot be the main reason for the low performance. We conclude that the low throughput was a result of the current version of the USB driver and the way of processing the high interrupt load on the system. Optimizing the process of handling the interrupts issued by the Network Interface Card will improve the performance of the network, which consequently will reflect on the IPsec performance [IVS09].

5.6 Summary

Our work is based on the observation that controlling the communications between different applications within a vehicular platform is a necessary prerequisite for containing an attack and preserving the security of the in-vehicle network. However, defining a static communications policy for a vehicular distributed system is both labor-intensive and error-prone and becomes more difficult if it is performed during the final integration stage. In this chapter, we propose a methodology supporting a *gradual* definition of the security policy, starting with the designer of the component, and then adapting and specializing the policy as the component is linked to other components and eventually integrated with the rest of the system. Additionally, we created a framework which allows communication policy to be deployed incrementally and ensures the integrity of this policy during the life cycle of software components.

The advantage of using a policy definition language for the expression of the vehicular communications policy is that we have a lot of expressive power which is reflected in our ability to *refine* and *extend* the original component policy during integration, while ensuring that the adaptations are consistent with the original policy. By creating a "chain of trust" from the original designer to the production system, we ensure the consistent application of the security policy without the need for manual intervention.

Maintaining the security policy enforcement mechanism on the operational vehicle does not impose an excessive performance burden, as demonstrated by the results from the preceding section. There we noted that the credential evaluation, whose overhead is shown in Figure 5.11, is done *once* per link setup. This operation usually takes place during power on, although, policy caching may enable results from previous configurations to be reused, further reducing the effort. Credential evaluation is not repeated unless the system is reconfigured. Moreover, re-keying the session key, which is done at regular intervals, does not require a credential evaluation operation. Even in cases in which a vehicle may face failure during operation and need to switch from a primary system to a backup system, the necessary policy may be already in place. We expect that reconfiguration will be done only when the vehicle is quiescent. Once the secure channels are setup, the overheads are similar to the statically configured case.

Also, our experimental result indicated that the increase in the latency value is small even after applying IPsec. However, since most of the security issues in the distributed embedded system communications are related to the lack of a proper authentication mechanism to prevent unauthorized parties from hijacking or sending false data, we believe that applying AH will solve all the issues related to the absence of data origin authentication and data integrity with acceptable overhead on the efficacy. Achieving privacy by using IPsec with ESP in such constrained resource systems will cause a significant overhead that might degrade the system capabilities in terms of throughput. We also believe that using more efficient encryption algorithm such as AES instead of a 3DES algorithm will improve the performance of ESP.

"An ounce of prevention is better than a pound of Cure"

Benjamin Franklin

6

Temporal-Based Intrusion Detection

Using the traditional mechanisms to eliminate vulnerabilities and to ensure security is insufficient by itself. As vehicle complexity grows, the likelihood of hidden vulnerabilities and threats increases. Therefore, it is critical to provide run-time mechanisms to detect the presence of malware and to stop the attackers before they manage to gain a foothold in the system. Using signature-based detection mechanisms is inadequate since it cannot keep up with the successive and unknown attacks which may target the vehicle during its operational life.

Anomaly-based detection techniques seem to be more convenient and capable of detecting any off-nominal behavior by the component, which could be caused by zero-day attacks. Anomaly-based mechanisms monitor the behavior of the component to identify any misuse that falls outside the predefined profile, which represents the nominal behavior of that component. The nominal behavior of a software component can be defined based on different component properties (e.g., system calls, memory consumption, etc.). In this chapter, we propose using the temporal specification of the observed task as a baseline to define its nominal behavior. Attacks such as a code injection or DoS attacks will usually cause a breach of this temporal specification, and thus it will be detected.

The most critical aspect when using anomaly-based techniques is ensuring the correct definition of the observed component's nominal behavior. Using inaccurate or the wrong behavior as a reference for off-nominal behavior detection may cause a high rate of false-positive and false-negative errors. One example of inaccurate nominal behavior definition is the use of safety-driven boundaries, which are used to define nominal behavior from a safety perspective to identify security-based nominal behavior.

In this chapter, we show how a safety-based temporal specification cannot be used to detect the malicious behavior of the observed component.

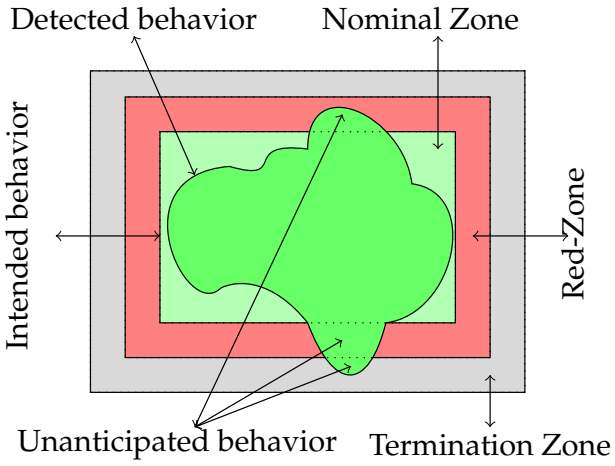


FIGURE 6.1: Red-Zone Principle

We also introduce the Red-Zone principle, which aims to relax of a predefined boundary in case they are very tight or to tight them in case they are over-approximated. Based on that principle, we identify the proper temporal thresholds which give the system the ability to predict the presence of malicious tasks. The system is then alarmed ahead of the actual temporal limits violation. This early detection of off-nominal temporal behavior provides a sufficient amount of time to initiate recovery actions. Based on that, we present a prediction scheme for Real-Time Operating Systems (RTOSes) where we integrate our defined configuration.

The rest of the chapter is organized as follows. In Section 6.1, we present the Red-Zone principle that we consider for intrusion detection and prediction as well as the required steps to adopt it. Section 6.2 provides a short background about real-time tasks and their temporal properties. Section 6.3 explains the use of these properties to support the intrusion detection process. Defining the prediction configuration for real-time tasks based on the Red-Zone principle is presented in Section 6.4. In Section 6.7, we provide explained how to implement the system. Later, we detail our prediction scheme and the implementation aspects in Section 6.8. Finally, Section 6.10 contains some concluding remarks.

Parts of this work have been published in [Ham+18; PH15b].

6.1 Red-Zone Principle

Every task, in a given system, is usually designed to accomplish a certain work and behave in a well-defined way. However, at run-time, a task may not exercise all the intended functionality and, in some circumstances, a task may stray outside its planned operational profile for many reasons;

- The task could be under an attack or it is already taken over and it is carrying out the attacker's commands.

- The task has encountered an error due to a hidden software bug as a result of insufficient testing.
- The task is affected by unforeseen circumstances in the system environment (e.g., a heat wave or exceptionally heavy rain).

However, straying outside the predefined policy is not always refer to a malicious behavior. The task may attempts to perform an action which although legal, is not permitted by the existing policy. This is usually the result of an incomplete security policy. This could be the case when the security policy is defined by using under-approximated or very limited rules as a result of incomplete test which did not cover all the implemented functionalities.

The security policy in many systems is defined as binary actions (authorized, not authorized). The system keeps a task running only if it behaves correctly by staying within its designated operational profile; otherwise, it will be terminated.

Terminating a perfectly functional component because of a violation, such as an overflow, may have spectacularly unintended consequences, as the crash of the inaugural flight of the Ariane 5 booster demonstrates [LL96]. Moreover, In the case of an attack, terminating the malicious task without any relevant information about the vulnerability may not be the right choice, as an attacker could use the same obscure vulnerability to break the newly instantiated tasks repeatedly. In addition, in the case of bugs, we need to identify the chain of events that led to the failure which will hopefully help us identify the bug in the design or the implementation of the system. Lastly, the off-nominal behavior may not be malicious after all.

Therefore, rather than terminating the task, we introduce the *Red-Zone* principle to permit the task to overrun its designed operational profile until an ultimate limit but in an observed mode. This window of observation is called the Red-Zone. Whenever the task exceeds the limit, it will be terminated, leaving behind it an audit trace with potential information about the vulnerability or the fault. Figure 6.1 illustrates the possible operational behaviors of a task. The four main areas represent: the intended behavior, the actual behavior, the Red-Zone, and the termination zone for a given task.

An important benefit of red-zoning is that, assuming the attack is detected, it allows an attack to be monitored in real time, while the subverted application cannot cause problems to the system. We call this an "instant honeypot" because the application ends up behaving like a honeypot [Pro03] at the moment it enters its red zone.

Another critical goal of red-zoning is to provide an early detection of off-nominal behaviors which will alert the system and provide the ability to respond in an effective manner. The effective response is necessary for safety-critical systems where losing or in-cognizant stopping of a subsystem may severely impact the overall system's behavior.

6.1.1 Adopting the Red-Zone principle

To adopt the Red-Zone principle, the following three-stage process needs to be accomplished:

- Establish the nominal behavior of the task.
- Formulate a security policy which defines the various execution areas for the task based on observations made in the previous step. Moreover, the security policy should determine the appropriate actions within each zone.
- Implement a framework to enforce the security policy while the state of the task is changing, reflecting the execution zones (nominal, red-zone, off-nominal).

In general, the nominal behavior of a program cannot be established using static analysis [AM14]. Looking at the source code, it is not easy to determine things such as: which parts of the code will be executed most frequently, how many elements will populate the data structures, which files will be opened, and so on. Consequently, we need to determine the program's behavior through dynamic analysis. Specifically, we need to execute the program many times, collect useful information from its execution, and analyze this information to determine its nominal behavior. Effective dynamic analysis requires that the program be exercised adequately during the data collection phase. For programs that come with a good set of test cases, executing the program with those test cases could be adequate. Although the precise number of required executions cannot be determined a priori, it is possible to use a control-flow or data-flow adequacy criterion to establish an upper bound on the number of program executions.

Many mechanisms were used to monitor the nominal behavior of a task based on different program properties such as power consumption [JMD04], system call distribution [Yoo+17], system calls sequences [Ahm+09], control flow graphs [BKM07], and so forth. Applying these mechanisms in the vehicular domain entails several challenges including the limited computing/energy resource of most ECUs. In this work we investigate another property. As we have stated in Section 2.1.2, modern vehicle contains safety-critical and non-safety-critical applications. One of the main characteristics of safety-critical applications is that they have strict temporal constraints. The timing behavior of a safety-critical system could be used by an anomaly-based detection mechanism to detect various attacks against such systems.

6.2 Properties of Real-Time Systems

In real-time systems the correctness of the system depends not only on the functional results of the computations, but also on the time at which the results are produced [SR90]. A reaction that occurs too late could be useless or even dangerous. For example, anti-lock braking system (ABS) on a vehicle

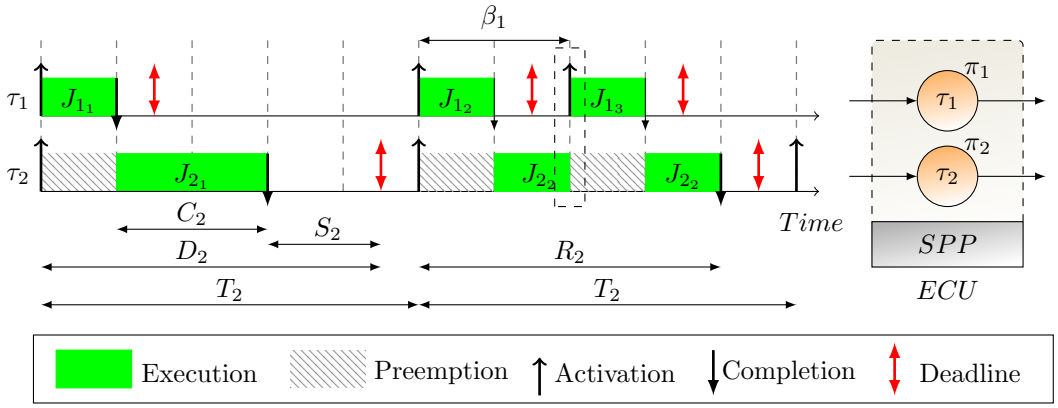


FIGURE 6.2: Timing properties of real-time tasks. Dashed rectangle represents the effect of SPP scheduling policy where τ_1 (has the highest priority) takes over the CPU previously allocated to τ_2 (has the lowest priority), thereby interrupting its execution.

need to react to the brake-pedal pressure and activating the brakes within a specified period; after this period, the vehicle may skid or crash into something. It is important here to point to a popular misconception that the real-time system is a system which performs very fast.

The performance of real-time systems has to be validated against timing constraints to guarantee that the results are available when they are needed. Timing analysis is often used in order to provide off-line safe upper bounds on the tasks execution and response times, in order to cover the set of all possible executions including corner-cases which are the smallest (best) and largest (worst) values that can be observed at runtime.

Before we go further, let us look in details at real-time tasks and their different properties and timing constraints:

6.2.1 Real-time Tasks

Each real-time task has number of execution requirements illustrated in Figure 6.2. Every real-time task τ_i is defined by a tuple $\tau_i = (C_i^+, D_i, T_i)$ where C_i^+ refers to the worst case execution time, D_i refers to the relative deadline, and T_i refers to the minimum interarrival time. Each task is assigned a *priority* π_i, if a task is activated regularly we call it a *periodic* task otherwise we call it a *sporadic* task and β_i is used to characterizing the minimum distance between two consecutive activation. Every instance of τ_i is called a *job* J_{ij}. When a job of τ_i is *activated* and becomes ready for execution, it occupies the resource (typically The CPU) for a duration no longer than C_i^+ of execution. The longest execution time (i.e., C_i^+) can be derived using worst-case execution time analysis. The point of time when the task finishes its execution called *completion* time. The idle time between the completion of a job of τ_i and its deadline is called a *slack* S_i. Such a slack can be used for the computation of that job without causing it to miss its deadline [DTB93].

Real-time task executions are constrained in time to guarantee the correctness of the system. This time constraint is the relative *deadline* D_i which is the maximum allowable time for a given task to complete its execution. The consequences of missing the deadline can be used to distinguish between two types of real time tasks:

- ***Soft* real-time tasks:** Missing a deadline of such tasks is tolerable and it may cause a degradation in quality of the service. In most cases, non-safety-critical tasks are considered as soft real-time tasks.
- ***Hard* real-time tasks:** Missing a deadline of such tasks is intolerable, and it could lead to catastrophic situations. All the safety-critical tasks are hard real-time tasks.

Similar to that, a (sub-)system of the vehicle is considered as a *Hard* real-time system when it contains at least one *Hard* real-time task. On the other hand, when all the tasks are *Soft* real-time task, we refer to the (sub-)system as a *Soft* real-time system.

6.2.2 Worst-Case Execution Time

Timing analysis computes for each task the Best-Case Execution Time (BCET), we refer to it as C_i^-, and the Worst Case Execution Time (WCET), we refer to it as C_i^+, guaranteeing that the execution time of the task will be confined to these bounds. The derivation of the WCET of a task is the first concern in the performance validation of real-time systems. WCET represent an upper bound of the execution time of a given task when it is executed on a particular hardware.

The WCET of a task depends both on the program flow, which includes loop iterations and different function calls, and on architectural factors such as pipelines and caches [EES01]. WCET can be calculated via various techniques includes static [Wil+10; HF04], measurement-based [BFM97; WM01], or hybrids of them [BCP02]. Using static techniques, see Figure 6.3, do not require the execution of the task on a real hardware. Instead, it requires examine the source code of the task to determine its structure and program flow, and combine this information with an abstract model of the hardware architecture to obtain upper bound for the execution time of the task. The WCET estimate by searching for the path with the longest execution time and calculating the execution time of this path. It is very important to emphasize that static methods guarantee safety by producing bounds that the execution time will not exceed these bounds.

6.2.3 Worst-Case Response Time

When real-time tasks are mapped to the same ECU, they share and compete for the same hardware resource (e.g., CPU). The execution of these tasks is controlled by a scheduler which decides at each time which task can be executed on the CPU. Different scheduling policies are adopted by the scheduler to arbitrate between the various tasks [MA05; Bal+98]. Static Priority

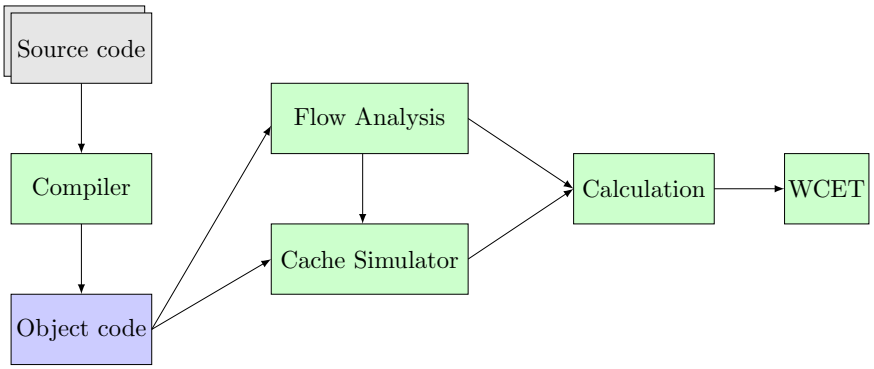


FIGURE 6.3: WCET Analysis based on [EES01]

Preemptive (SPP) scheduling [Aud+95] is well-known scheduling algorithm which by adopting it, the scheduler will always allocate the resource to the ready job of the highest priority task (see Figure 6.2), therefore, a lower priority task may be preempted by a higher priority one.

The *response time* R_i then characterizes the time which spans between the activation of a job and its completion. Response time analysis [Sha+04] is used to determine the worst-case response time (WCRT), it computes an upper bound on the response time of a task and validate it against its deadline D_i. The busy-window analysis is a well established technique for computing the WCRT. It computes the maximum processing time $B_i^+(q)$ which is necessary to process q activation (jobs) of a task τ_i. $B_i^+(q)$ under an SPP scheduler is defined as follows:

$$B_i^+(q) := q.C_i^+ + \sum_{j \in hp(\tau_i)} \left\lceil \frac{B_i^+(q)}{T_j} \right\rceil .C_j^+ \tag{6.1}$$

In a periodic system, $\left\lceil \frac{B_i^+(q)}{T_j} \right\rceil$ is the maximum number of jobs from interfering tasks τ_j during a time window of size $B_i^+(q)$. Note that, the computation of the busy-window is a fixed point computation as $B_i^+(q)$ appears in both sides of the equation, which can be solved iteratively, starting with $B_i^+(q) := q.C_i^+$. The computation of $B_i^+(q)$ is proven to converge for some $Q_i \in \mathbb{N}$ [Aud+95].

R_i^+, the WCRT of task τ_i, is computed as follows:

$$R_i^+ := \max_{1 \leq q \leq Q_i} \{B_i^+(q) - (q-1)T_i\} \tag{6.2}$$

Figure 6.4 presents a motivational example which we will use through the next sections of the chapter. This example represents a hard real-time system includes three periodic tasks τ_2=(1,4,4), τ_3=(1,5,5), and τ_4=(2,8,8) ordered from the highest to the lowest priority. The system also contains one sporadic task τ_1 with execution time $C_1 = 1$ ms and $\beta_1 = 2$. The task τ_1 has the highest priority but is rarely activated. By using Equation 6.2, we can compute the WCRT for every task of the system.

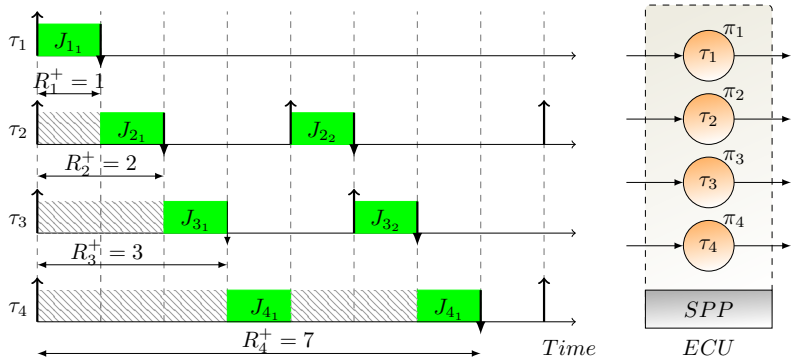


FIGURE 6.4: WCRT calculation for τ_1, τ_2, τ_3 and τ_4

Assumption: In this work we consider that the studied system is a hard real-time system with a uni-processor and SPP scheduling policy.

6.3 Time-Based Intrusion Detection Approach

As the vehicle increasingly becoming more interconnected with the Internet; the different hard real time (sub-)systems within it become an attractive target of various attacks. However, these systems have the advantage that they are predictable by design, as they should comply with temporal constraints.

Therefore, from the security point of view, we refer to any malicious activity which alters the predefined temporal behavior of a real-time task and causes it to miss its deadline as a temporal attack. The temporal attack could be caused by deliberate attacks, such as injection code attack when an attacker tries to exploit existing vulnerability (e.g., buffer overflow [And72]) to execute injected or preexisting malicious code to break the system or to force it to do malicious actions [Kos+10]. The deadline miss may be the objective of the attack (e.g., DOS), or a side-effect of the attack as the process now needs to carry out both its normal tasks and the actions of the attacker. If the resources allocated are tight, the extra effort will result in slower response time and potentially missed deadlines.

In hard real-time systems, a task is considered as off-nominal whenever it violates its temporal constraint (i.e., deadline). Therefore, an intuitive approach is to use the deadline as a demarcation line which indicates the off-nominal temporal behavior of the task. However, using the deadline miss as a sign to predict the off-nominal behavior is insufficient for many reasons. Firstly, waiting for the task to exceed its deadline in order to trigger an alarm about the off-nominal state is useless, especially in critical-safety hard real-time systems, since it is too late for the system to recover, we refer to this as *belated detection*. Secondly, the deadline miss of one task may cause a cascading deadline miss for lower priority tasks as shown Figure 6.5, the deadline miss of τ_3 causes τ_4 to miss its deadline too. Finally, the use of deadline-based monitoring may not always identify the off-nominal task correctly. Indeed, the interference on a high priority task may lead lower priority tasks to miss their deadline. In this case, the deadline-based detection will report an issue

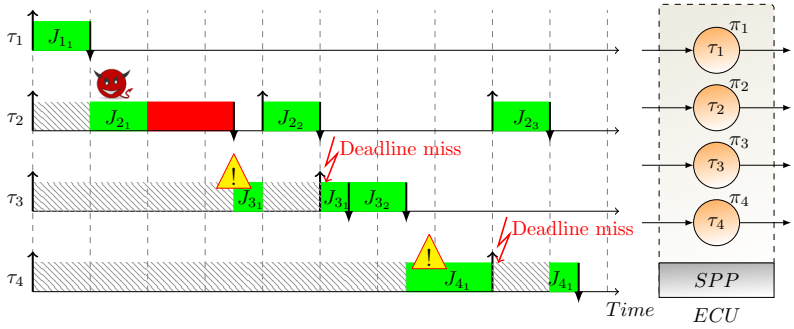


FIGURE 6.5: Using the WCRT and deadline miss as an indicator of compromise could lead to false identification of malicious tasks.

in lower priority tasks without any sign about the initial misbehaving of the high priority task. In Figure 6.5, monitoring system will report both τ_3 and τ_4 as malicious tasks since they miss their deadline although τ_2 is the one should be identified since it is the actual task which causes this issue.

AUTOSAR, for instance, also recognized the problems associated with deadline-based monitoring approach. Therefore, it chose to monitor the execution time of tasks rather than the deadline to ensure the time protection. This mechanism's goal is to prevent fault propagation by killing the task which exceeds the WECT [BFT09]. However, as mentioned previously this extreme solution may not be suitable for hard-real-time systems, as it may cause the system to enter an unstable state. Zimmer et al. [Zim+10] use the WCET timing bound to detect code injection attacks by comparing the WCET value with the elapsed time along the return path. Petal et al. [PP08] proposed an approach to monitor the execution time of the running tasks. They use a dedicated security processor to supervise the application processors. They change the binary file of the task by inserting instructions at the start and the end of each block to instrument the execution time for each block. At run time, the security processor will determine whether a given block takes longer time than its defined WCET.

Therefore, defining other reasonable temporal boundaries to delimit the off-nominal behavior of real-time tasks is required, in order to prevent the propagation of the temporal constraints violations between tasks and to detect early enough the presence of a malicious task in the system. We aim to define these boundaries by adopting Red-Zone principle.

6.4 Red-Zone Boundaries for Real-Time Systems

We extend the use of the Red-Zone principle to establish appropriate prediction configurations which can be adopted by different real-time systems. These configurations define the monitoring boundaries which are used later by the prediction scheme to detect the anticipated off-nominal tasks behavior. The adoption of the Red-Zone principle is achieved by an off-line analysis for each real-time task to define:

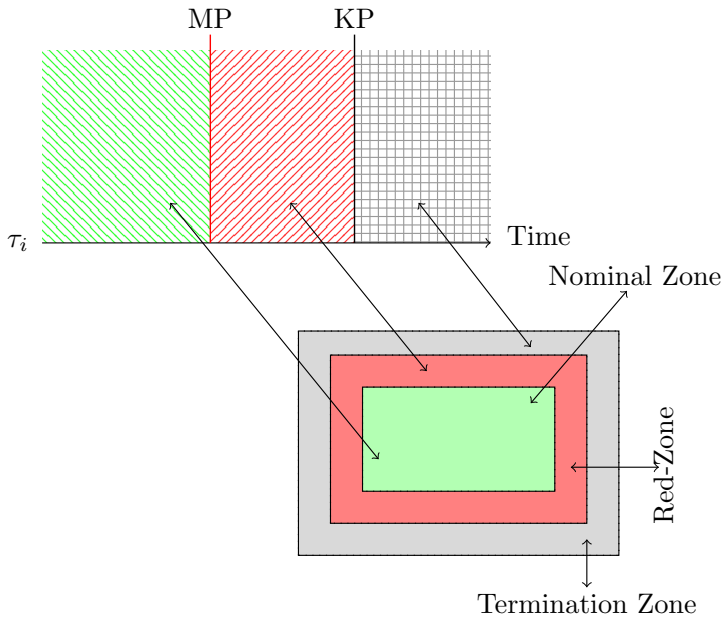


FIGURE 6.6: linking the Red-Zone areas to temporal line of the real-time task.

- The **nominal temporal behavior** of each task.
- A **Monitoring Point (MP)**: it is a point in time which represents the border line between the nominal temporal behavior and the potential off-nominal one, as in Figure 6.6. The temporal behavior of a task becomes suspicious whenever this point is passed.
- A **Killing Point (KP)**: it is another point in time (see Figure 6.6); the temporal behavior of a task becomes certainly off-nominal whenever this point is exceeded. At this stage, a task should be either terminated, the recovery policy should be applied, and/or the control may be transferred to back-up system.

Two key features determine the efficacy of any proposed mechanism for off-nominal behavior detection, namely the False Positives Rate (FPR) and the precision values. FPR, which is defined in Equation 6.3, identifies the number of suspicious tasks which complete their jobs before the killing point over the number of all monitored ones. An effective prediction techniques should ensure a minimum value of FPR.

$$FPR = \frac{\#\{R_i \mid R_i \geq MP \quad \& \quad R_i < KP\}}{\#\{R_i \mid R_i \leq KP\}} \tag{6.3}$$

Precision, which is defined in Equation 6.4, used to determine the accuracy of the off-nominal behavior detection scheme. An accurate prediction technique should ensure a maximum value of precision. it is defined as follows:

$$Precision = \frac{\#\{R_i \mid R_i \geq KP\}}{\#\{R_i \mid R_i \geq MP\}} \leq 1 \tag{6.4}$$

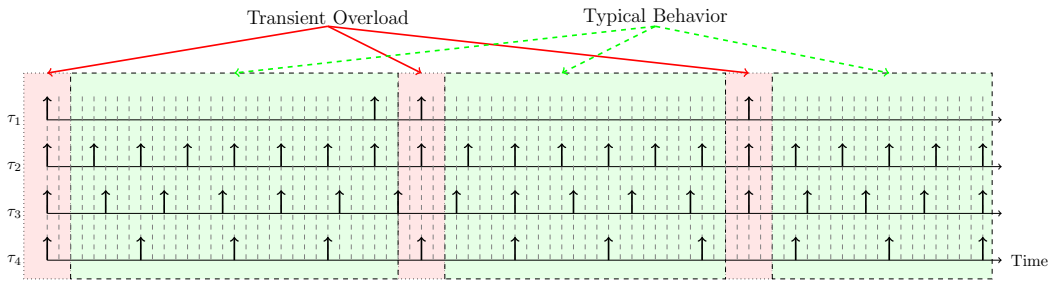


FIGURE 6.7: Execution trace of τ_1, τ_2, τ_3, and τ_4.

As mentioned previously, the Red-Zone principle defines a time zone where we can start monitoring the system in order to safely detect a possible malicious behavior and act on-time accordingly, before we consider the extreme solution of terminating a task or a system.

The Red-Zone can be defined in different ways taking into account the timing properties of real-time systems. In our work, we use the response time (Section 6.5) and execution time (Section 6.6) properties of real-time tasks in order to define boundaries (i.e MP and KP) for the red-zone detection mechanism. Note that in both Equations, 6.3 and 6.4, we have used the response time R_i to compare it with both MP and KP. In case we define the boundaries based on the execution time of the task, we replace R_i with the execution time (C_i).

6.5 Response Time-Based Prediction Configuration

As we described in section 6.4, we need to define the response time-based temporal nominal behavior of the hard real time task. Consequently, we determine the MP as well as KP.

6.5.1 Defining Temporal Nominal Behavior and MP

In real-time systems, the WCRT can be used in order to characterize the system. An intuitive approach to designate the Red-Zone is to use WCRT as a MP. Starting the monitoring at the WCRT allows detecting attacks that solely push the response time of the system to be larger than the worst-case. However, waiting until the WCRT to start tracing the task is, in many cases, too late. Allowing the response time to exceed the WCRT may cause a cascading deadline miss for lower priority tasks as shown Figure 6.5 where the response time of τ_2 has exceeded R_2^+ which causes both τ_3 and τ_4 to miss their deadlines sequentially.

Furthermore, in a real-time system, which contains periodic and sporadic tasks, the WCRT does not reflect the ordinary nominal behavior boundary of the task. WCRT analysis takes into account the worst case scenario where the maximum interference (i.e load) is considered. However, in many cases,

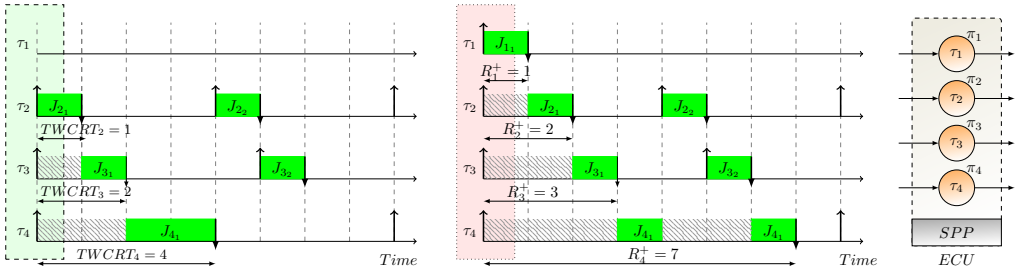


FIGURE 6.8: The TWCRT and WCRT for the τ_2, τ_3, and τ_4 .

in particular, in the presence of a transient overload (i.e., sporadic tasks), the frequency of activation of these tasks is very low and therefore the worst-case scenario with maximum load may never happen, but is nevertheless considered, which results in very large bounds on the response time. Figure 6.7 shows the execution trace of the 4 tasks in our example for a period. It is clear that most of the time the temporal transient overload does not exist (as in the dashed, green rectangle).

Typical Worst-Case Analysis [QHE12] (TWCA) was introduced in order to take into account the activation frequency of sporadic tasks and therefore define a much smaller/tighter bound on the response time accompanied by the frequency at which this bound might be violated based on the frequency of the sporadic load [HR95]. Consequently, the purpose of the typical worst-case is to get as close as possible to the most probable case to occur at run time leading to typical (and not worst-case) performance. In other words, it is used to define the nominal behavior boundary of the task.

By applying TWCA analysis for every task τ_i in the system, we get:

- a typical worst-case response time bound $TWCRT_i$ which could be used as MP_i for the task τ_i , along with
- for every $k \in \mathbb{N}^+$, a bound on the number of executions of τ_i (let us say m) which may have a response time larger than $TWCRT_i$ in a window of k consecutive executions. We refer to it as *m out of k*. This bound represents the maximum permitted FPR_i of our prediction schema for the task τ_i.

Indeed, we can define the nominal temporal behavior of task τ_i based on this analysis that the nominal response time of task τ_i is between $]0, TWCRT_i]$.

TWCA refers to the response time of the task τ_i without the transient overload (i.e., ignoring the sporadic tasks) as a *typical* worst-case response time $TWCRT_i$ for this task. Figure 6.8 shows the two relative response times; left part of the figure shows the $TWCRT$ of the tasks while the right part shows the $WCRT$ which we already calculated. From the same figure we can see the difference between the $TWCRT$ for τ_4, which is equal to $4ms$, and the $WCRT$, which is equal to $7ms$.

By unfolding multiple executions of the periodic tasks (i.e., τ_2, τ_3, and τ_4), one can observe an upper bound of how many times out of the observed executions when those tasks may coincide with τ_1 and have a response time

larger than its $TWCRT$. For example, by considering the execution trace presented in Figure 6.7 and by calculating the response time of the three periodic tasks during the tracing period, we find that out of 20 consecutive executions of τ_2, only 4 observed R_2 were longer than $TWCRT_2$. And, out of 16 consecutive executions of τ_3, only 2 observed R_3 were longer than $TWCRT_3$. For task τ_4, out of 10 consecutive executions, only 2 observed R_4 were longer than $TWCRT_4$.

Figure 6.9 represents the relation between the FPR_i and the defined bound (i.e., *m out of k*) for τ_i. In our system, FPR_i dose not refer to a probability. At the start of monitoring, FPR_i has a high value which will decreased by the time of detecting cases where the a response time larger than $TWCRT_i$. After reaching the bound m, any response time larger than $TWCRT_i$ will consider malicious. However, there is no guarantee that no one of the m cases was not a result of the attack.

6.5.2 Defining KP

After defining the MP, we need to determine the other bound which enclose the Red-Zone area (i.e., KP). In *hard* real-time systems, a task becomes malicious whenever its response time exceeds its deadline. Therefore, deadline is potential candidates to be used as KP. But, using the deadline as KP is not suitable; keeping the task running until its deadline will cause a cascading deadline miss of the low priority tasks, as we have seen before.

Another proposal is using WCRT of the task as a KP. However, it is known that monitoring the task will introduce some overhead, i.e. Monitoring Overhead (MOV). Therefore, using WCRT as a KP may cause killing (in the case that the security policy requires the termination of the task when it passes its KP) a benign task since the WCRT of this task was calculated without considering such an overhead. Thus, we defined another time point for each task to represent its KP, called the Early-warning Deadline (ED).

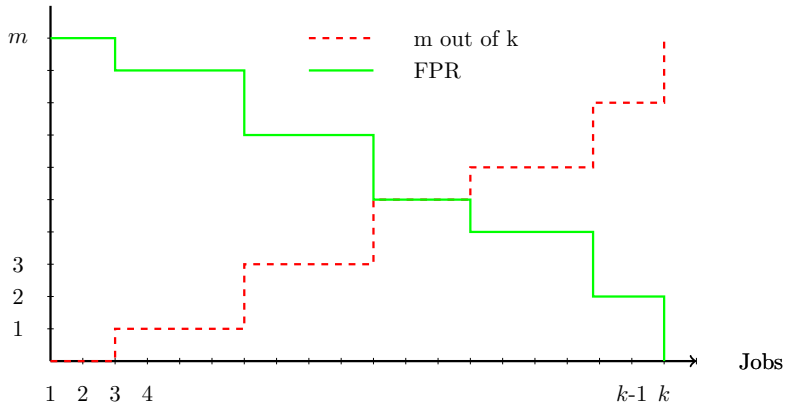


FIGURE 6.9: The relation between the detected FPR_i and the m *out of* k for task τ_i

Definition 6.5.1. ***Early-warning Deadline (ED):*** *ED_i of task τ_i is the maximum allowable time for τ_i to complete its execution without causing the lower priority hard real-time tasks to miss their deadlines.*

The ED_i of any task τ_i represents the worst case response time in the existence of the monitoring overhead once for each one of the higher priority tasks. To calculate the ED_i we need to use the available idle time (i.e., slack) without jeopardizing the hard timing constraints.

ED_i of each task τ_i is defined as:

1.
$$WCRT_i < ED_i \leq D_i \tag{6.5}$$

2.
$$ED_i = C_i + MOV_i + \sum_{j \in hp(\tau_i)} MOV_j + \sum_{j \in hp(\tau_i)} \left\lceil \frac{ED_i}{T_j} \right\rceil . C_j \tag{6.6}$$

With initial value $ED_i = C_i$ and under the conditions that

$$\sum_{k \in H} MOV_k \leq \min_{l \in H} S_l \tag{6.7}$$

where S_l represents the slack of task τ_l, H represents the set all hard real-time tasks in the system, $\forall i \in H$: $D_i \leq T_i$.

The monitoring overhead MOV_i is different for every task τ_i and is proportional to the size of its Red-Zone.

$$MOV_i = (WCRT_i - TWCRT_i) \times f \tag{6.8}$$

where, f is a factor which represents, for each task, the introduced overhead per unit of time.

Figure 6.10 shows the the ED of each task in our example based on Equation 6.6. Note that for the sake of simplicity calculation of ED we assumed that the overheads sum is equal to the minimum slack (i.e., $\sum_{k \in H} MOV_k = S_{min} = 1$).

Figure 6.11 illustrate the proposed prediction configuration which defines the Red-Zone for hard real-time tasks, and the relation between these points and the existing temporal constraints.

6.6 Execution Time-Based Prediction Configuration

Defining the time behavior of each task in the safety-critical system is a mandatory requirement. In avionics systems, as an example, the standard RTCA DO-178C [Inc11] requires determining the $WCET$ for each task in the airplanes under-development. Using static analysis methods to compute the $WCET$ will produce an over-approximated bound of the actual observed

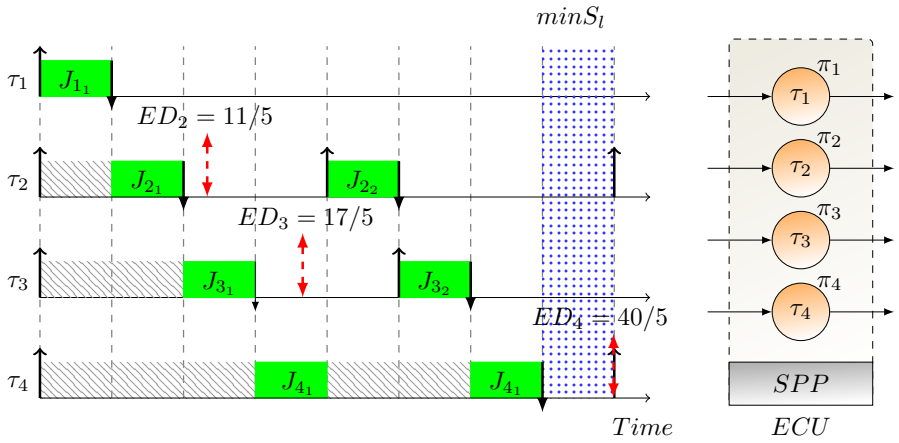


FIGURE 6.10: The ED for the τ_2, τ_3, and τ_4 .

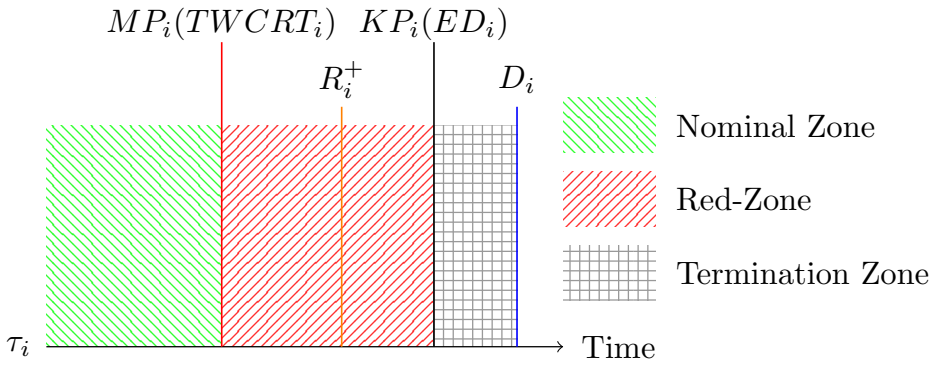


FIGURE 6.11: Temporal prediction configuration for hard real-time systems.

value. Therefore, such worst-case scenario is rare to occur in real life, although it is possible. The computed *WCET* traditionally refers to the maximum consumed time in the longest execution path. The use of such over-approximated thresholds for detecting the off-nominal behavior of a real-time task, as proposed in [LSL15], may give false sense of security. Indeed, attackers may not always exploit the longest path. Violating the temporal bound of arbitrary paths may not be detected while they are less than the *WCET*.

Moreover, system could contain tasks which have multiple run-time behaviors. Such tasks behave differently based on the system modes (e.g., an initialization mode, a recovering mode, etc.). The off-line calculation of the *WCET* using the static methods considers only the worst scenario from all modes. Therefore, using a single *WCET* is not suitable to detect the malicious behavior of a task with different running modes.

Calculating a bound on the typical case for the execution time similarly to the response time (i.e., most probable average behavior at run-time) is very difficult using statistical analysis methods. Therefore, in order to define a suitable execution-time based prediction configurations, we propose the use of measurement-based (i.e., dynamic) methods or hybrid methods [Wen+05]

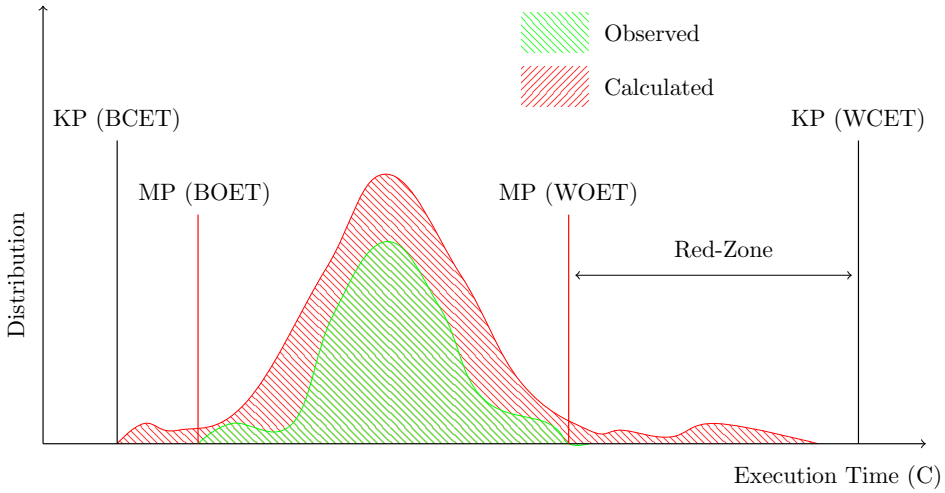


FIGURE 6.12: Execution time distribution.

to compute a measured *WCET*, where we execute the task, in each mode if it is a multi-modes task, for a sufficient number of times on the given hardware or a simulator for different set of inputs. Then, we derive the Worst and Best Observed Execution Times (we refer to them as *WOET* and *BOET* respectively) from the measured times. Figure 6.12 shows the relation between the two produced values. The Red-Zone can then be defined by using the *WOET* as a *MP* and the *WCET* as a *KP*. The introduced *MOV* of the monitoring is covered as we assume that the *WCET* approximation is large enough to include it. Another Red-Zone area can be defined between the *BCET* and *BOET* to detect the malicious behavior of a task when it completes its job in this interval. However, we are not able to trace the task in this case since we do not know the completion time in advance. But we could, based on the security policy, ignore the produced result, use the recovery one, and/or monitor the next job of the task in the next period.

6.7 Design and Implementation

Our prediction configurations are defined for the system with temporal constraints. Such systems require a dedicated operating system to run it. RTOS is a particular kind of operating systems which are designed primarily for systems with real-time requirements. RTOS uses scheduling algorithms, such as SPP, to ensure deterministic behavior of the different tasks in the system. Each task is represented by its Task Control Block (TCB). RTOS uses the task's TCB to maintain the properties and internal states of that task.

The common strategy is to integrate the implementation of the monitor in the source code of the tasks, calculate the consumed time by each function of the monitored task, and compare it with the saved temporal thresholds. Such an approach suffers from the late detection of the temporal violation. Not only that but it also has other drawbacks:

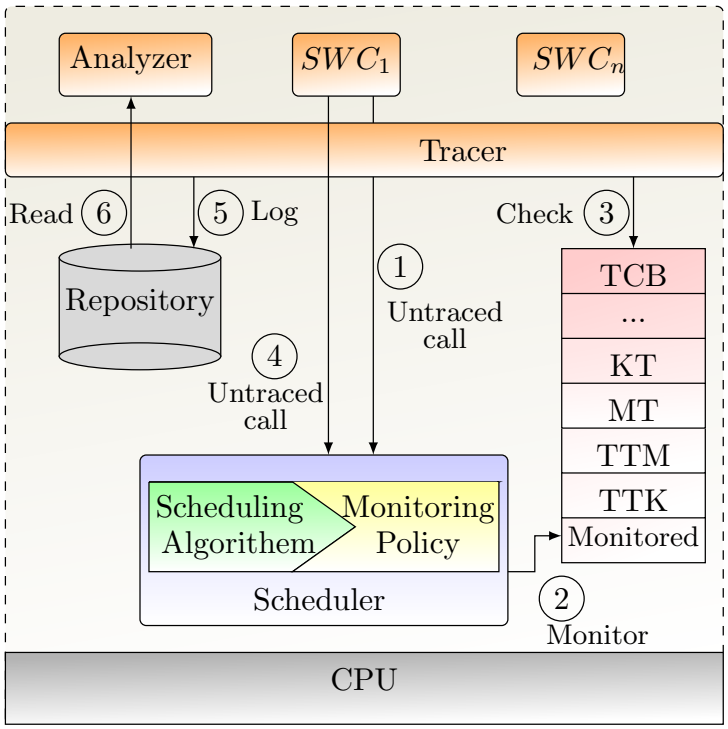


FIGURE 6.13: The prediction scheme and the steps to monitor the temporal behavior of tasks.

- Firstly, most of the safety-critical applications are certified. Moreover, most of these tasks do not contain the monitoring code before they have been certified. Adding monitoring instrumentation for these tasks will cause the loss of the code integrity and require issuing a new certificate.
- Secondly, since the monitoring code is not included from the start of the task implementation, the computed time constraints do not consider the produced overhead by the monitoring. The newly added time overhead for the task may cause the violation of the complete schedulability analysis test.
- Finally, the monitoring instrumentation will be part of the task itself; they share the same address space; consequently, an attacker who could manage to take over the task will be able, naturally, to manipulate the monitoring behavior.

Therefore, it is more efficient to implement the monitoring scheme into the kernel (i.e. in the scheduler). Since implementing the prediction mechanism inside the kernel does not require modifications to the source code or object code of the existing tasks. Moreover, by placing the monitoring code in the kernel we isolate and protect it from the vulnerable code in the user space tasks. Finally, the existence of the monitoring code in a single location will give us the ability to calculate the newly introduced overhead efficiently; we do not need to recalculate the WCETs for every task in the system.

We propose a prediction scheme, shown in Figure 6.13, compatible with the RTOSes. Figure 6.13 illustrates the logical ordered steps to predict the

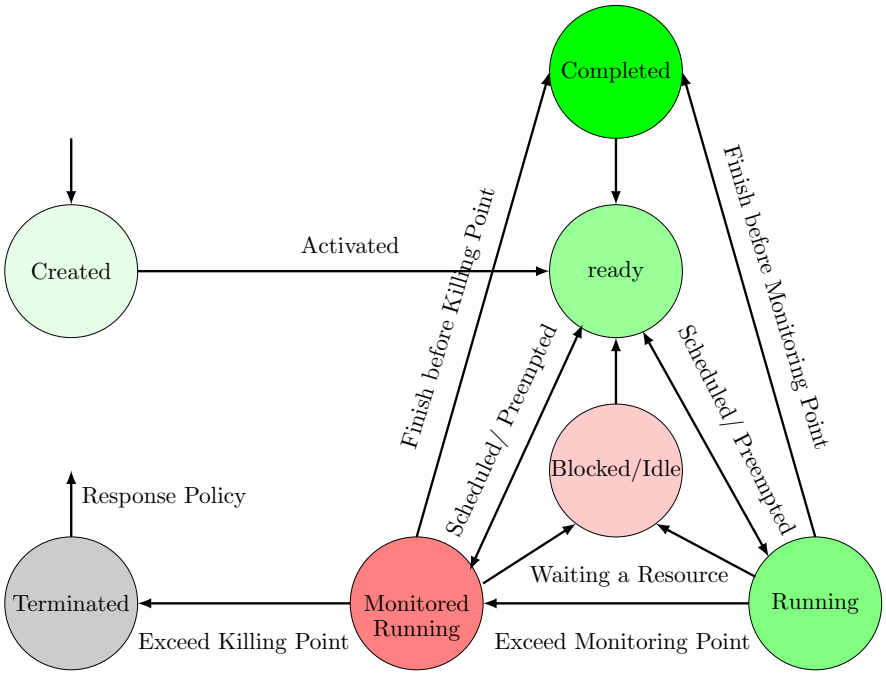


FIGURE 6.14: Monitored Running state and its relation with other states.

off-nominal behavior of a task and trace it. The schema contains three main parts: the new task state (i.e., Monitored Running) and the modified task TCB, monitoring policy which is integrated into the scheduler, and the tracer which logs the task activity. In the remainder of this section, we elaborate on these parts in more detail.

6.7.1 Task States

Basically, each real-time task is designed to have a recurrent nature. During its life cycle, it can be in one of three different states: either it is *Ready* to run, it is already *Running*, or it is *Blocked* (i.e., it is waiting for some resources, or it is *Idle* waiting for a specific timeout). In our proposed scheme, we introduced another state called *Monitored Running* state as shown in Figure 6.14. Whenever the task exceeds its monitoring point, it will enter to the *Monitored Running* state where it is running under observation. During its running in this state, the task could be blocked waiting for a resource, or another task could preempt it. It could complete its execution before the killing point (KP) or it may breach its KP which would force it to be terminated (i.e., based on the applied security policy).

The implementation of such state could be accomplished in many ways. The first option could be by adding a new queue, called monitored queue, to the system similar to the existing queues (i.e., ready queue or blocked queue). In this case, the scheduler needs to consider the new queue when it decides to pick up the highest priority task to run it. The other option could be by modifying the task's TCB and add a new field to identify the state of the task. This Boolean field will be checked whenever the task exceeds its monitoring

point. Using the queue option could be more efficient if we want to know all monitored tasks since they are all gathered in one place, without the need to search in all existing tasks.

6.7.2 Monitoring Policy

Whenever a new task is created, the defined prediction configurations for this task should be added to its TCB. Therefore, we modified the TCB to include another four fields as shown in Figure 6.13. The first field is called Monitoring Time (MT), which holds the value of the monitoring point. The second field, called Killing Time (KT), contains the value of the kill point. The third field is called Time To Monitor (TTM), and the last one is called Time To Kill (TTK). At the initiation time, the TTM and TTK fields contains the values MT and $(KT - MT)$ sequentially. We used the TTM and TTK fields to save the consumed time during each phase (i.e., nominal and Red-Zone) whenever the task is preempted by another task with higher priority.

All these values are used by the integrated monitoring policy as described in algorithm 1. Whenever the task is scheduled to run, the TTM value is loaded into a timer (line 5). The task is kept executing normally without any tracing (line 7-13) while its execution time does not exceed the MT (i.e., the timer is exhausted). In the mean time, if a task with higher priority shows up, the timer stops and its remaining balance is saved into TTM field of the task TCB (line 9), and then, the context switching takes place. Whenever the timer reaches its limit (i.e., task's execution time passed the MT), the state of the task is changed to "monitored" and an interrupt is caused (line 14-15). The same procedure will be applied when the task is running under the "monitored running" states (line 16-27).

6.7.3 Tracer

The tracer is implemented as a thin layer which wraps all the system calls. The tracer logs the calls and parameters only if the calling task is in the monitored running state. This will reduce the overhead of excessive logging of the benign tasks' activities. However, the malicious code could start ahead of the monitoring point. Consequently, the logged information could not be sufficient for detecting the reason for the off-nominal behavior. In this case, the security policy could include a rule to trace the task from the start in the next period.

The logged information can be used either in an off-line or on-line analysis. On-line analysis requires introducing a separate process called "Analyzer". Finding the temporal slot to integrate such a task in the existing system is studied in detail in [Has+16a; Has+16b; Ham+17]. Deciding the temporal type (i.e., periodic, or sporadic) of this task is quite important. Defining the Analyzer as a periodic task may introduce the problem of determining the period of this task. Using a short period may add redundant overhead when the Analyzer is running and the repository is empty. On the contrary,

Algorithm 1 LoadTask: using the prediction configuration while loading the ready task. CntaxtSW: context switching and saving the monitoring values.

```
1: procedure LOADTASK( )
2:     TIMER timer
3:     TASK cur_task                                   ▷ current task
4:     if (cur_task.TCB.monitored = False) then
5:         timer.value ←cur_task.TCB.TTM
6:         start timer
7:         while (timer.value >0 & cur_task not Done) do
8:             cur_task runs without tracing
9:             if (Preemption) then
10:                stop timer
11:                CntaxtSW (timer.value)
12:            end if
13:        end while
14:        cur_task.TCB.monitored ←True
15:        GoTo 17
16:    else
17:        timer.value ←cur_task.TCB.TTK
18:        start timer
19:        while (timer.value >0 & cur_task not Done) do
20:            cur_task runs in monitored running state
21:            if (Preemption) then
22:                stop timer
23:                CntaxtSW (timer.value)
24:            end if
25:        end while
26:        Kill cur_task or apply recovery policy.
27:    end if
28: end procedure
29: procedure CNTAXTSW(integer t)
30:    if (cur_task.TCB.monitored = False) then
31:        cur_task.TCB.TTM ←t
32:    else
33:        cur_task.TCB.TTK ←t
34:    end if
35:    save cur_task.TCB
36:    load the preempting task.
37: end procedure
```

the repository could be flooded, especially in an embedded system, if the Analyzer has a long period.

Therefore, we decided to run the analyzer task as an sporadic task under the assumption that the system is schedulable. The analyzer will be activated

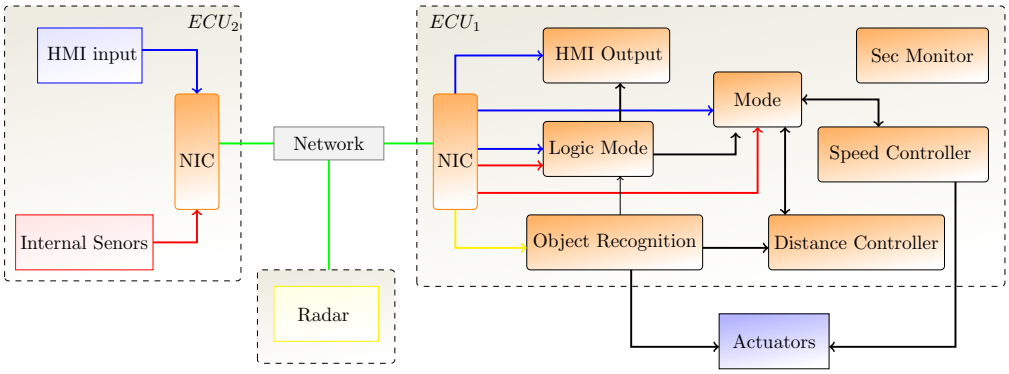


FIGURE 6.15: ACC software components based on [Han+04]. The color of arrows within ECU_1 refer to the source of the information which has the same color and mapped to ECU_2.

whenever the tracer logs information to the logging repository. The implementation of the analyzer is however out of the scope of thesis.

The designed schema can be modified to predict the off-nominal behavior using the response time-based configuration by using a hardware timer for each real-time task and without the need to save the consumed time for each stage when the context switching is taking place.

6.8 Evaluation

To evaluate our approach of adopting the Red-Zone on a real-time system, we use the ACC use case which was explained in Section 3.2. Figure 6.15 shows the various software components which collaborate to achieve the ACC functionality besides security monitoring component (Sec Monitor). All these software components are mapped to the same ECU (i.e., ECU_1), and require information delivered by other components within different ECUs and sensors (e.g., Radar and internal sensors). The Radar sensor is used to monitor the road ahead and inform the Object Recognition task. As long as there is no object ahead, ACC maintains the speed set by the driver through HMI input. Otherwise, ACC reacts based on the speed of the detected object. In case that a head vehicle slowed down, Speed controller reduces the speed of the car by releasing the accelerator and/or by activating the brake control system. If the road becomes clear again, ACC accelerates to reach the desired speed. In both cases, HMI output is used to inform the driver about the current speed and other statues. The vehicle's driver can, at any time, disable or enable the system which is captured by the Logic mode component.

Table 6.1 illustrates the real-time properties of ACC tasks which are mapped on the ECU1. Within this system, there are two sporadic tasks (i.e., τ_0 and τ_1) and the rest have periodic activation. Note that in this system the deadlines of the periodic task are defined to be equal to the periods ($D_i = T_i$). The tasks are ordered in the table from the highest priority to the lowest one. We are interested in comparing the defined MP with the calculated WCRT

TABLE 6.1: WCETs and Deadlines of software components of ACC system. Values for τ_2-τ_6 are based on [Han+04].

Task	Task Name	T	WCET
τ_0	Sec Monitor	150	4
τ_2	Speed Controller	20	5
τ_1	NIC	10,200	6
τ_3	Distance Controller	100	20
τ_4	HMI Output	100	2
τ_5	Mode Logic	100	1
τ_6	Object Recognition	100	30

TABLE 6.2: Red-Zone response time based prediction configuration and FPR.

Task	MP	WCRT	ED	FPP
τ_0		4		
τ_2	5	9	9,33	2%
τ_1		16		
τ_3	30	51	52,75	8%
τ_4	32	53	56,17	8%
τ_5	33	54	58,58	8%
τ_6	73	94	100	8%

for each critical task and see how early we could alarm the system. Moreover, we are concern about how often that our approach could indicate a false off-nominal behavior of each monitored task (i.e., FPR value).

In Table 6.2, we show the computed bounds on each task, namely: the worst-case response time (WCRT), the typical worst-case response time which is considered as the monitoring point in our scheme (i.e., MP), the killing point (overall monitoring overhead is considered equals to minimum slack), and the false positive ratio. By comparing the WCRTs with the MPs we observe that we alarm the system (putting the task in the monitoring mode) early enough - in this example the MP/WCRT is ranging between most 55 to 78% - with a very low FPR (in average 8%). A one has to be aware that the killing points KP are fair enough to let the task finishing its execution in case of false positive monitoring (normal execution) and at the same time to prevent any deadline miss for the lower priority tasks which may cause a degradation in the QoS provided by the system.

One other important concern in the proposed scheme is its performance overhead. Our monitoring algorithm is integrated to the system scheduler, thus, it is applied for each hard and critical task in the system. The added overhead of this modified scheduler comes from the use of the timers and the required time to save its value later during the context switching. Another overhead comes from the tracer; This overhead directly proportional to the type of the logging mechanism and the logging details. In our current proposal, we traced only the application's system calls.

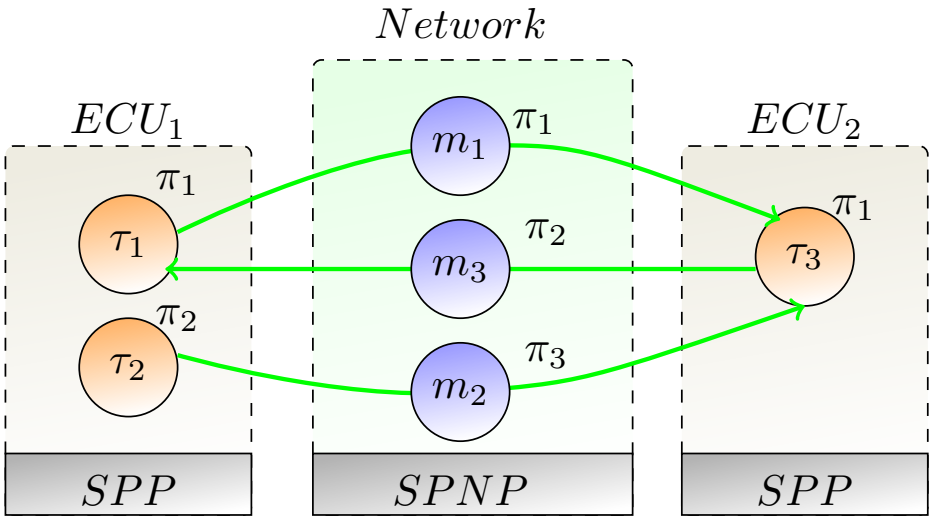


FIGURE 6.16: Moving from host based toward network based IDS.

6.9 From Host to Network Based IDS

As we have seen in subsection 2.3.2, many proposed network-based IDS have used the period of the transmitted messages over the bus as a reference to detect any off-nominal behavior such as injecting malicious messages or dropping messages. But it is crucial to know that the in-vehicle messages do not have consistent timing intervals. The changes in regular driving operation can change these timing intervals [You+19]. Also, sporadic messages, by its nature, do not have such fixed period. These two cases make many of the network-based IDSs, which depend on the period, not reliable.

In the previous sections, we have shown how we used the TWCA to create temporal boundaries for the different hard real-time components in the system by considering the sporadic components. These defined boundaries are more suitable for detecting the security attack, which affects the temporal behavior of the system tasks. Figure 6.16 shows how we could similarly look at the network resources as a single ECU which hosts the hard-real time components. But, instead of the software components, we have the messages which are transmitted by these components. Each one of these messages also has its priority which is different from the one that applications that emit this message have. Also, instead of using SPP, we consider the scheduling algorithm for the network to be non preemptive (i.e., static priority non-preemptive (SPNP) scheduling policy). By considering that, we can apply the TWCA on the different messages, as in [Qui+14], will provide us with TWCRT boundaries for these messages which could be used as an input for the temporal-based network-based IDS.

6.10 Summary

The early detection of the off-nominal behavior in safety-critical systems is an important goal. It gives the system the ability to recover in an efficient manner and prevents the fault propagation. In this chapter, we proposed a prediction configurations for real-time tasks with temporal constraints to predict the off-nominal behavior of these tasks. We used in particular the TWCRT to define a bound which represents the least privilege temporal bound for each real-time task. Whenever the execution (or the response) time of a given task exceed these bounds, an alarm is triggered to notify the system about a potential off-nominal behavior which may be caused by the execution of malicious code. Such configuration allows the task to execute outside its designated profile under carefully controlled conditions.

Moreover, we presented a prediction scheme compatible with the RTOS. The scheme used the prediction configuration to monitors the taskś behavior and provides relevant log activities of the task during the relaxation of the security policy (i.e., during the Red-Zone). Our approach depends on tight temporal bounds which makes it applicable to be adopted by existing schedulable hard real-time systems. It is well known that determining precise prediction bounds is always a challenge. However, the accurate definition of the Red-Zone bounds will improve the efficacy of the prediction scheme. As a future work, we could use different methods (e.g., statistical analysis method) to define Red-Zone bounds and compare the efficacy of using the produced bounds with our current system.

"It's not what happens to you,
but how you react to it that matters"
Epictetus

7

Towards a Vehicular Intrusion Response System

Recently, developments within the vehicular domain have made the modern vehicle a network comprised of a multitude of embedded systems communicating with each other, while adhering to safety-critical and secure systems specifications. Many technologies have been integrated within modern vehicles to give them the capability to interact with the outside world. These advances have significantly enlarged the attack surface. We have already had numerous instances of successful penetration of vehicular networks both from inside and outside the vehicle. To deal with these attacks, many intrusion prevention and detection mechanisms were implemented inside vehicular systems, as we have shown in chapter 2. Despite the adoption of all the proactive security mechanisms mentioned in the previous chapters, there is no guarantee that the system becomes secure and that no attacker can strike it. Having a system with absolute security is not achievable, and the proof is that despite all these mitigation mechanisms, attackers were able to exploit these vulnerabilities and mount severe attacks against a vehicle [Kos+10; Che+11; Rou+10].

Since a vehicle is a safety-critical system, the response to an attack is as pivotal as the detection of the attack itself. When an attack against a component is detected, one way to respond is to restart the affected component [Str+09], in case the resulting failure was a transient one. High system resilience and safety are two fundamental aspects that need to be always guaranteed by several response strategies, which the security policy of the vehicle implements. These response strategies are implemented through an IRS. Generally - and even more so in the vehicular domain - IRSs have received less attention and research efforts compared to IDSs [SBW07], until now. In this chapter, we try to investigate the main requirements and challenges that face the adoption of IRSs in the vehicular domain. In addition, we propose

an intrusion response framework for in-vehicle systems based on the RedZone principle (which was discussed in Section 6.1). Parts of this work have been published in [HTP19b; HTP19a].

The rest of the chapter is organized as follows. In Section 7.1, we present some main taxonomy related to the IRS in general. In Section 7.2, we present the challenges that IRSs face in a vehicular context and the requirements to design such system. In Section 7.3, we present our proposed intrusion response framework for in-vehicle systems. The development of this IRS and the different related aspects are explained in Section 7.4. In Section 7.5, we discuss the development process as well as the proposed responses using the automated obstacle avoidance use case. Finally, we summarize our chapter in Section 7.6.

7.1 Intrusion Response Systems

As the advancement of technology offers capabilities which result in attackers at every level getting more competent and effective, attacks have become more elaborate. Therefore, we need to establish an adequate level of security in software systems. Complete system security of a system is unfeasible and so it becomes imperative to detect a situation where an attack might be unfolding and take action to respond to it. Consequently, intrusion prevention mechanisms (firewalls, cryptography, access control, etc.) alone are not sufficient to mitigate these attacks. IDSs were widely developed to detect, analyze and report intrusions in a computing system. Whenever a task behaves off-nominally or violates a predefined security policy, the IDS considers this task as a malicious one. Some IDSs have already implemented limited static responses, such as generating an alarm or report [KV02]. However, with increasing levels of attack complexity and targeted domains, more comprehensive response strategies are required. These strategies could be implemented through an IRS.

As the advancement of technology offers capabilities which result in attackers at every level becoming more competent and effective, attacks have become more elaborate. Therefore, we need to establish an adequate level of security in software systems. Complete system security is unfeasible and it thus becomes imperative to detect a situation where an attack might be unfolding and take action in response. Consequently, intrusion prevention mechanisms (firewalls, cryptography, access control, etc.) alone are not sufficient to mitigate these attacks. IDSs were widely developed to detect, analyze and report intrusions in computing systems. Whenever a task behaves off-nominally or violates a predefined security policy, the IDS considers this task to be malicious. Some IDSs have already implemented limited static responses, such as generating an alarm or report [KV02]. However, with increasing levels of attack complexity and targeted domains, more comprehensive response strategies are required. These strategies could be implemented through an IRS.

Authors in [SBW07; SS+12; Ina+16; Foo+08] have surveyed the existing IRSs. They have proposed a taxonomy of these systems according to different characteristics:

Nature of response: This feature determines the activity of the selected response. IRSs can issue a response which aims to limit the effect of the attack and minimize further damage (i.e., active response). Another IRSs could respond only by notifying (or alarming) the system of the existing attack (i.e., passive response).

Time of response: This aspect determines when the responding action should take place. IRSs either activate the response after the occurrence of the attack (delayed response) as in [Lew+01] or take action before the attack has affected the system resources (proactive response), such as [Foo+05].

Degree of automation: This characteristic defines how the response occurs. Some IRSs require the interference of the system administrator in order to apply the predefined response (manual response), while others are totally independent and do not require any human interaction to react to an intrusion (automatic response).

Response selection: This is used to distinguish between the different IRSs based on the way these systems choose the applied responses. Most of the IRSs use a fixed predefined response for a certain alert (static response) [Loc+05; Ryu+03]. A number of IRSs chooses the response based on attack metrics, therefore the response for the same attack could be different from one instance to another (dynamic response) such as [Bal+03; TK02].

Cooperation capability: This characteristic determines how the IRS acts during the attack. This includes whether it works alone and applies its chosen response locally (stand-alone system) such as [Bal+03]], or it collaborates with other coexisting IRSs to apply the selected response locally and globally (collaborative), such as [Lew+01].

7.2 Vehicular Intrusion Response System

The vehicle, which is considered as a safety-critical system, has its own special properties and restrictions which limit the adoption of the existing IRS of the other domains. Automotive systems are different from the conventional networks from many perspectives. In this section, we explain some of these challenges which affect the design of any intrusion response framework for the in-vehicle system.

7.2.1 Challenges

- **Highly interconnected architecture:** As we mentioned in chapter 2, a vehicle includes large numbers of ECUs running software supplied by

different vendors with widely varying degrees of security awareness. In order for the vehicle to function properly and safely, these ECUs must exchange data, and therefore many of them share the same communication bus. Unrestricted interaction among those ECUs puts the entire system in danger. An attacker could launch a stepping-stone attack, in which they compromise a less important ECU with weaker security (e.g., the entertainment system), in order to gain control of a more crucial one (e.g., engine ECU) [Kos+10]. Consequently, any response or mitigation mechanism should consider the highly interconnected and distributed nature of a vehicular environment.

- **Limited resources:** The computational power of the embedded ECUs inside a vehicle is far lower compared to the capabilities of computers and servers in traditional networks. This limits the possibility of porting strategies which are already in place in those networks, because the limited-resource ECUs will be unable to cope with them.

- **Ever-changing scenery:** Many people think that the vehicle is a fixed environment in which no changes need to take place after vehicle production. But that is not entirely true, especially with current and future vehicles. Cars are designed to work for an average of 12 years [Mar16]. So, a car in its lifespan may have updates to both its hardware and software components. This means that the rules of the security policies in an IRS cannot be static, but must be defined as dynamic and changing in order to accommodate the behavior of the newly introduced components.

- **Autonomous and semi-autonomous nature:** In a traditional IT environment, a human administrator would be expected to approve and apply responses. Nowadays, some might consider the driver as the administrator of the vehicular environment. However, for safety reasons, the driver's attention must not be diverted while driving. Moreover, a driver may not have the technical knowledge to address an attack underway or the required response, so must not deal with such decisions. Furthermore, in the case of an autonomous car, these actions will be performed automatically, without any input from the driver.

7.2.2 Requirements

The existing IRSs which are deployed in other domains deal with only a subset of the aforementioned challenges. Therefore, based on both the challenges of the vehicular domain and general IRS taxonomies (see Section 7.1), we explain here the desired properties for any proposed IRSs for in-vehicle systems. The proposed IRSs should be:

Proactive: The IRS should predict an attack on the system and not wait until after the attack takes place to take an action. Early prediction provides the system with a sufficient amount of time to respond in an effective

manner. However, false positives are of notable concern here. Unfortunately, as the security constraints are tightened, the likelihood of false positives increases.

Active: The proposed system should react in a way that mitigates the damage caused by the attack and prevents its propagation to other subsystems. In addition, it must consider issues such as containment, continued availability, interaction with other subsystems and, in certain cases, latency. These requirements are in many cases in conflict with each other; for example, security and availability often work against one another [TGK00].

Proper: The IRS should react properly, based on the type and target of the potential attack. In a case where the IRS detects that a component is acting off-nominally, it responds to that behavior by terminating (killing) the running process and executing a new instance. However, such a response is not preferable because it may cause an accident. Moreover, because of the lack of direct interaction with the driver to solve the security issue, terminating the malicious task without any relevant information about vulnerability may not be the right course of action, as an attacker could use the same obscure vulnerability to break the newly instantiated task repeatedly.

Heterogeneous: The IRS should be designed to respond in multiple ways, including a fully automated response. When a task is terminated, regardless of the cause of termination (security related or otherwise) the system should respond to the failure by (i) using a redundant component to provide the functionality that is lost with the demise of the failed task, (ii) determining whether the system can continue running without this functionality (fail-continue), or (iii) forcing the system into a fail-safe condition, i.e. it goes into a mode where the safety of the vehicle and passengers is assured even if the system can no longer continue to operate. Although current regulations ensure that safety decisions must only be taken by the driver [HKD09], the situation may need to be changed to reflect the rise of autonomous and self-driving technology. In the meantime, a semi-automated response which requires partial interaction with the driver could be adopted.

Dynamic: The IRS must be able to differentiate its response to the same attack based on several factors, such as contextual information during the vehicle's different operational modes, the attack severity, the subsystem targeted and other dependent systems. However, creating static security rules or a decision table for all these different specifications is not a trivial task. Therefore, the use of a dynamic security policy to express these parameters could be a preferable solution [Deb+07; HNP17].

Deactivable: Any security measure employed to counter a malicious attempt comes at a price. As such, the IRS introduces overhead on the system's performance, based on the type and duration of the response.

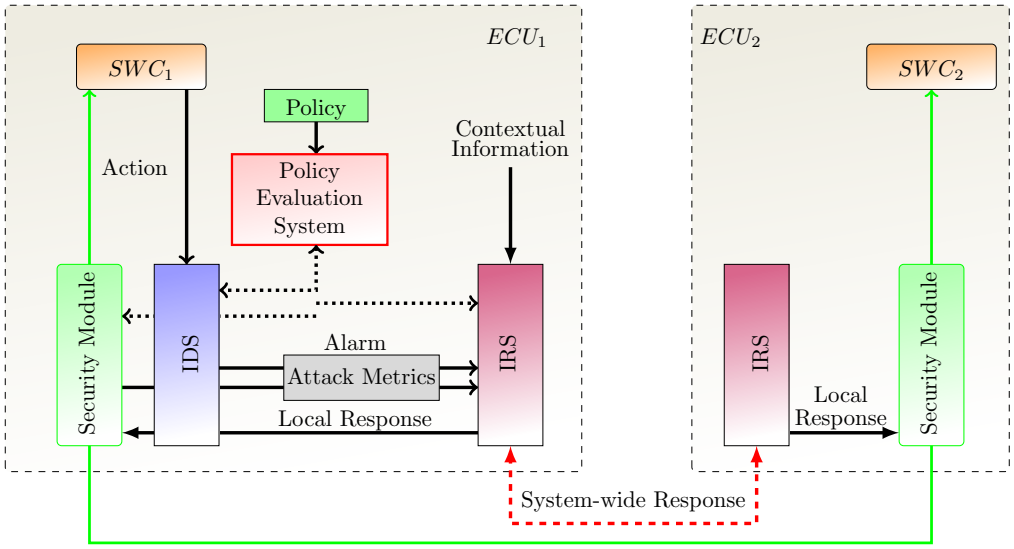


FIGURE 7.1: Intrusion response system and its relationship with IDS and security module.

To minimize this overhead, any proposed IRS should be able to deactivate the response mechanism when specific, predefined conditions specified within the security policy are met [Kan+13].

Distributed: Each ECU should be protected by its own IRS. When a component is potentially under attack, the corresponding IRS detects the malicious attempt and decides locally how to deal with it. Although this decision may address the attack against the specific component, it may not be optimal for the overall system. Each individual component should be able to implement a response strategy locally which, however, must be defined for the system as a whole.

Responsive: Activating the response action should take place immediately after the attack is detected. If a significant amount of time passes, the damage might already be done, negating the existence of the IRS. Therefore, the proposed IRS should evaluate the security policy as well as any other inputs with minimal delay.

Low-cost: As mentioned before, the resources of the embedded ECUs are limited. Therefore, the IRS should operate in a cost-effective manner and not consume system resources more than necessary, thus avoiding a situation in which other components are prevented from functioning properly.

7.3 Proposed Intrusion Response System

Figure 7.1 shows our proposed multilayer security framework for every single ECU within the vehicle. The figure shows the different layers of protection against intrusions and how they collaborate with the IRS, which represents the last layer of this framework. IRSs are used to determine the action

needed when a security violation is detected either by the security module (explained in Section 5.3) or the IDS. The main aim of this layer is protecting the vehicle from entering an unstable state as a result of the discovered attack.

The IRS requires inputs from the different system components (such as the properties of the detected attack or off-nominal behavior) which is delivered by the security module or the IDS as a part of the alarm message sent to the IRS whenever an incident is detected. The security policy is also needed for this layer since it includes the possible responses to different attacks. Contextual and operational information is also gathered by a detective service and delivered to the IRS when it is required. The IRS contains a repository which is used to store the collocated characteristics of attacks (i.e., aim, scenario, severity, etc.). This information is used as feedback or input for the next threat modeling process.

7.3.1 Red-Zone as an IRS

Improving the response strategies of any security mechanism is dependent on early detection (or the early prediction) of a potential attack. Early prediction provides the system with a sufficient time to initiate recovery actions to respond effectively. Our strategy to solve this conundrum involves using the Red-Zone principle which we introduced in a previous chapter (see Section 6.1).

Figure 7.2 shows how we can use the Red-Zone principle as a base to develop an IRS and how it could be used to manage the interaction with the IDS. It also shows how, with time, the security attack turns into a failure which requires the interference of safety countermeasures (i.e., the Fault Tolerance System (FTS)). Based on the Red-Zone principle, the IDS is used to monitor and evaluate the behavior of the different vehicle software components within each ECU, based on predefined security policies. Each policy specifies the correct behavior of these components. Any violation of this policy is used as a prediction of a potential security breach. At this point, the task enters the Red-Zone area and becomes suspicious. The activation of any response mechanism is triggered as soon as the system receives an alarm about a detected potential attack from the IDS. During the Red-Zone window, the IRS is responsible for activating local and system-wide response strategies.

7.3.2 Local Response Strategies

Local tactics are used to respond to an attack against a component and other components on the same ECU, as well as the whole ECU itself. The local response to the *malicious component* can be:

- **Suspicious / Malicious Operational:** Although the task is considered malicious, the IRS leaves the task running until a certain limit. While it is executing, the IRS starts a series of possible responses, e.g., the component could be isolated and its communication with the other components on the same ECU or with other sub-systems becomes very limited

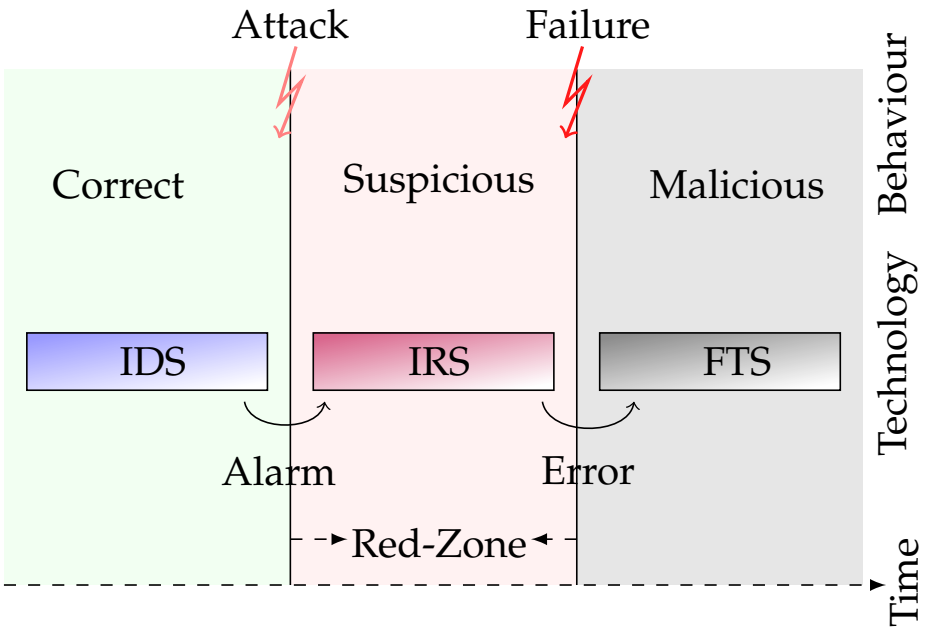


FIGURE 7.2: IRS based on the Red-Zone Principle

and begins to be traced. The logged information can be used later to design a proper mitigation mechanism. Moreover, the trust level of the component is degraded, so other components become more cautious about the information delivered to/from this component.

- **Suspicious / Malicious Silent:** The IRS kills the process whenever it exceeds the ultimate bound since the component will suffer from a security failure at this point [Avi+04]. Although the IRS may stop the component at the end, which could be the goal of the attacker seeking to cause a DoS, the IRS has plenty of time to activate other response strategies prior to this point. On the other hand, the IRS can implement a response which ensures that a new healthy instance of the malicious component is initiated with its entire state reverted back to a specific point at which it was executing correctly, before killing the malicious one.

All the above-mentioned tactics for the infected component affect the *other components on the same ECU* as well. For example, some less critical software components which are mapped to the same ECU on which the malicious component is running may have to become silent/inactive to save resources for the system to audit and recover.

The IRS applies very firm restrictions on the *ECU which hosts the malicious component*. This could include performing more security auditing operations on the communications of this ECU to prevent any stepping-stone attacks. If the attack was propagated across the whole ECU, total isolation of the ECU could be adopted.

It is important to note that the location where the IRS is placed as well as the targeted system architecture play a significant role in determining the

proper reaction. For example, the authors in [FL15a] propose disabling the targeted ECU instead of the one hosting the malicious component. The reasoning behind such a selection is that it is hard to identify the origin of incoming messages in the case of a CAN network.

7.3.3 System-wide Response Strategies

System-wide responses are implemented by other IRSs on different dependent ECUs within the vehicle. Such responses are activated as soon as a notification from the IRS where the suspicious component exists, is initiated. We can view these response strategies as an extension of the local ones. Here we summarize some of the possible responses based on the locally applied response:

- **Reallocation**: As we mentioned before, IRSs aim to ensure the availability of the different components by initiating at least one new healthy instance of the malicious component (more than one replica can be used at the same time). The place where the new instance is mapped is critical for the other dependent system components. If the new replica is mapped to a new ECU, other components that depend on the misbehaving one need to communicate with the newly initiated component instead. This requires the use of a new communication configuration to adjust the reallocation.
- **Degradation and Propagation** occur as a result of stopping certain non-critical software components on the malicious ECU to support the adoption of local responses. The same strategy has to be applied to other interrelated ECUs which include components dependent on the silent ones. In the same manner, if the trust level of the data provided by the malicious component is reduced, the trust level of the other elements which use this data needs to be adjusted accordingly.
- **Combination**: The IRS could adopt a combination of the aforementioned tactics to achieve the best possible protection.

7.3.4 Intrusion Response Exchange Protocol

The proposed Intrusion Response eXchange Protocol (IRXP) is used to exchange the attack information and the system-wide response strategies among the different IRSs. Figure 7.3 represents the different steps of IRXP. The protocol is activated whenever the IDS or security module detect a violation of the security policy of any software components on the hosted ECU. The IDS or security module passes the information of this violation as an alarm to activate the local IRS. This alarm includes detailed information about the attack (or the suspicious action). This information includes the targeted component, the attack details, and attack detection time. All these information are transmitted using a predefined format similar to the alarm format proposed in [DCF07].

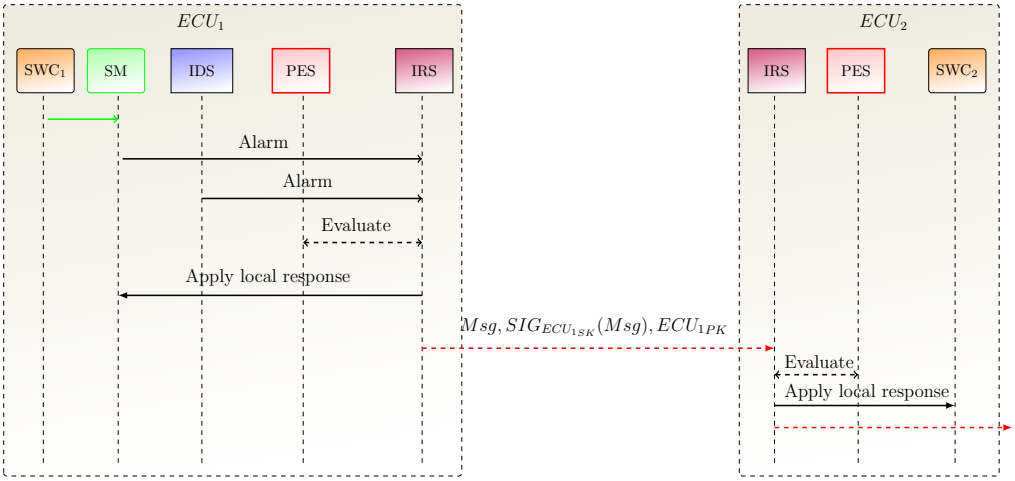


FIGURE 7.3: Intrusion Response Exchange Protocol

This information is used by the local IRS to determine the required local response for this attack. PES is used to support identifying such responses based on the attack metrics as well as the local system policy. The IRS takes these responses and sends them to the suitable component to be executed. For example, it could be a new communication rule which needs to be added to the decisions repository of the security module. Later, these attack metrics can be customized and sent to the other IRSs hosted by other ECUs which could be affected by the local response. Besides the basic information about the attack, the applied local responses are also propagated to ensure the adoption of the system-wide response. we refer to these information as a message (Msg)

The protocol uses similar mechanisms to these ones used during the initiation phase of establishing a secure link that we have explained in Section 5.4.1. In order to prevent any malicious third parties from manipulating the transmitted message, the IRS uses the ECU's private key to sign the message (in our case IRS uses ECU_{1SK}). Also, a monotonic counter value can be used to prevent replay attacks by ensuring the IRS response freshness. IRS on the remote ECU (i.e., ECU_2) uses the received public key of the other ECU (ECU_{1PK}) to validate the message which will be used to feed PES (on ECU_2) with the required information to determine its local response. At this point, the IRS on ECU_2 repeats the steps that the IRS on ECU_1 has already performed.

7.4 Vehicular IRS Development

Developing the response strategies to cover the large number of distributed components is a difficult task, requiring detailed knowledge of possible interaction paths and the dependencies among all those components, as well as the security threats that could target them and possible attack scenarios. When considering an evolving system, which should be update-able (whereby component interactions may change), this task becomes even more

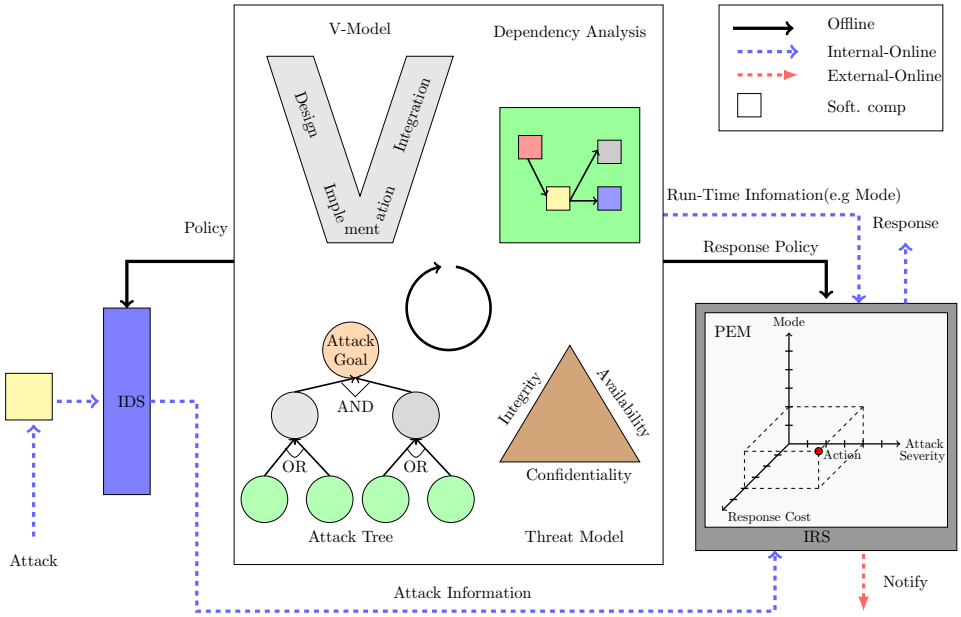


FIGURE 7.4: General Scheme of IRS Development

difficult. The planning for the response strategies has to be well integrated throughout the design and life-cycle of the system, i.e., it should be part of the different design stages of the V-model.

In this section we show how to integrate the development of the IRS within the entire development process of the vehicle system. In addition, we discuss the various aspects of the system which affect the design and implementation of the IRS. Figure 7.4 shows the general diagram for developing the IRS. In this section, we explain the different aspects which need to be considered during the development of the different responses.

7.4.1 Threat Model

In Chapter 4, we explain how threat modeling is used to describe and classify security threats which affect the vehicle system. Moreover, we detail how it provides significant information about the threats that could help to safeguard the target system. We add here that threat modeling can be used as a significant resource to develop effective response strategies against any attacks.

During the design stage of the V-model, security requirements are defined to address all existing vulnerabilities identified by the threat model. At the same time, the response actions for violating these requirements are defined from an abstract viewpoint. At a later stage (i.e., implementation and integration) the response actions become more detailed to reflect the final system architecture and ensure compliance with the initial actions.

While adopting our SAVTA threat model, a response for each violation of the security requirements (i.e., CIA^3) has to be planned. For example, violating the *integrity* of a software component may require reducing the trust level of that component. Violating the *integrity* of the RTE, which could be

detected using a secure boot, may require fixing this issue before letting the vehicle move. In such a case, we face a conflict with the *availability* requirement for each component (or at least, safety-critical ones). Such conflicts need to be considered and solved by proposing a convenient response to loss of *availability* of these components.

7.4.2 Attack Tree

Attack trees are used within our SAVTA threat model to detail how the attacker could achieve his goal (i.e., attack paths). The attack tree is created off-line for each system asset of the SAVTA threat model (e.g., hardware, software, data and surrounding environment). Then, the defined attack trees are used to evaluate the security risks, calculate the probability of a successful attack, and measure the defense (local response) cost for each proposed response [Edg+06]. Calculating these metrics depends on different aspects, as we have discussed in Chapter 4.

Based on the attack information which is detected by the IDS or the security module, a system can determine the goal in the attack tree. This gives it the ability to determine the attack's severity, which is used as an input for the IRS to choose the appropriate response, similar to [Foo+05].

7.4.3 Dependency Analysis

Another important aspect of implementing the response mechanism is to determine all dependency relations between other components and the malicious one. We call a task dependent on another task when it uses a service provided by that task. These relationships can be denoted in a dependency tree [TK02].

Then the direct and indirect dependencies are specified. Moestl et al. [ME15] have proposed a cross-layer dependency analysis to detect dependencies between the different components across the different architectural model layers in safety-critical systems such as vehicles. By defining the dependencies, the IRS on the local platform can notify all other dependent subsystems about the attack, and thus response mechanisms on other platforms are also deployed. In addition, the system-based response cost can be evaluated by using dependency analysis. A mechanism to ensure that the system will not enter an endless storm of responses is required. Entering such a situation may end up stopping the vehicle from functioning.

7.4.4 Security Policy

The security policy is used to define the response of the system when a task enters the Red Zone area, the conditions under which a Red-Zoned task may return to its normal state and, finally, the response to a task breaching its ultimate limit.

LISTING 7.1: Security policy with the different Red-Zone principle areas

```
if (property == nominal)
  do (allow)
else //Red–Zone
  if (property == suspicious)
    if (attack_severity== high)
      if(mode == startup)
        do(response)
  else // malicious, failure
    do(response)
```

Listing 7.1 shows the structure of the security policy. Based on a specific property of a component (such as execution time, power consumption, system call distribution or bit-rate of message exchange), the boundary of the Red Zone is determined in the policy and the response action within each zone is defined. Other properties such as the system operational modes (e.g., vehicle in startup mode) as well as the attack severity could affect the choice of response action. All this information is collected while the system is running and evaluated using the Policy Evaluation System (PES) to choose an action dynamically.

7.5 Use Case: Intrusion Response for Obstacle Avoidance System

We will use the automated obstacle avoidance system, which is introduced in Section 3.2, as a use case to explain how we could develop intrusion response strategies. Figure 7.5 shows part of the final integration of that system on the actual ECUs. One important functionality which is needed for the autonomous vehicle to avoid obstacles is localization functionality. This functionality gives the vehicle the ability to determine its position with respect to the environment. The main information resource for this functionality is provided by the GPS. We want to focus on this functionality to show how we could develop response strategies when it is targeted by a security violation, and discuss all the aspects presented in Section 7.4.

In the first stage, the security requirements for the relation between this functionality and others are defined: (a) availability and (b) integrity. Any violation of these requirements demands the adoption of the proper responses. Attack trees are developed to identify all the attack scenarios which could violate these two requirements. Attacks such as a spoofing data attack, jamming attack, and many others are defined [Car03; Nig+12]. Based on the attacks details, detection mechanisms are defined and stated in the security policy to detect these attacks while the system is on-line.

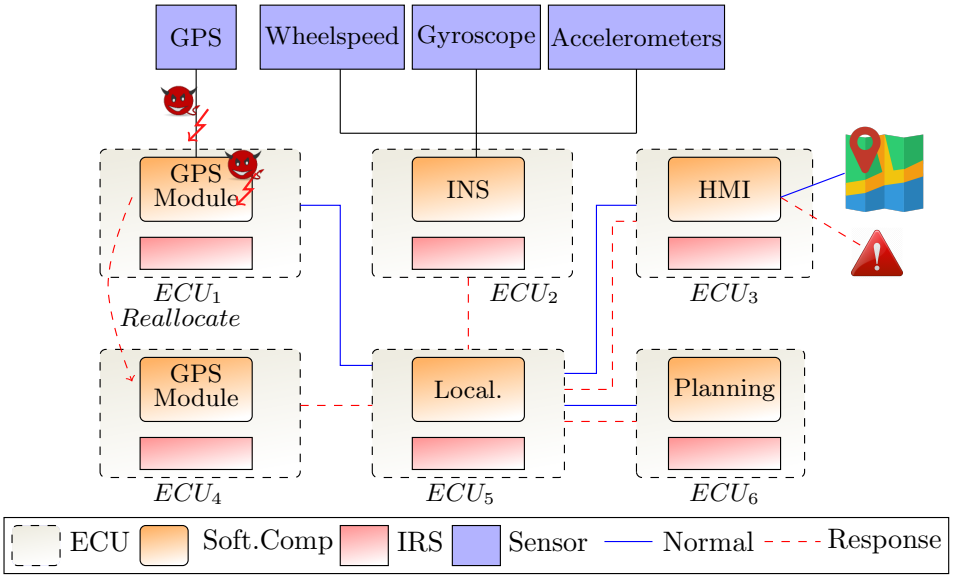


FIGURE 7.5: Response strategies in the presence of security attack on localization functionality of automated obstacle avoidance use case.

The responses to these attacks also are defined. One response is receiving the service from another source. The Inertial Navigation System (INS) could be used as a temporary resource for providing localization information. Another response option is to reallocate the GPS module (in the case of a Receiver Software Attack [Nig+12]) either within the same ECU or on another one (ECU_4 on our use case). Both strategies require communication reconfiguration for the direct dependent components. Dependencies analysis shows that the localization component (*Local.*), which is mapped on ECU_5, directly depends on the GPS module. Therefore, the security policy related to *Local.* component on ECU_5 should enable two communication paths with ECU_2 or ECU_4 (based on the chosen response) whenever the *GPS Module* is behaving maliciously.

Another response could be disabling the link with ECU_1 to prevent attack propagation. The *HMI* component, which is mapped on ECU_3 and indirectly dependent on the *GPS module*, could keep displaying the given data on the map, but with an alarm signal activated to express the low level of GPS data trust, as shown in Figure 7.5.

7.6 Summary

We propose to use the Red-Zone principle as the basis for developing an IRS framework. Whenever a component is under cyber-attack, the IDS notifies the IRS to be activated. The IRS uses the Red-Zone time period to implement response strategies. Based on this design, our proposed IRS meets the **proactive** requirement. There is a period before the task reaches its ultimate boundary and causes any harm. How long this period lasts is based on the

selection of the boundary for the nominal behavior as well as the monitored property. In the previous chapter, we have shown how the temporal property can be used as an indication to predict an attack at an early stage, before violating the ultimate temporal boundary [Ham+18]. However, false positives can be a side effect when increasing the Red-Zone time window.

During this Red-Zone period, the IRS is able to activate a first level of response. We have already implemented many response strategies during the previous chapters (e.g., system notification). The IRS can, subsequently, apply a second set of actions such as placing the task under supervision. Other response options can be defined through the security policy. Another level of response can be applied when the task enters the termination zone (i.e., system failure). All these different response venues attest to how our system can be **heterogeneous** and **active** in complying with the respective requirements.

The proposed framework is integrated within each ECU in the system (i.e., **distributed**). IRSs are designed to use a predefined security policy in different strategies. We have extended the security policy, which was used in our previous work [HNP17] to define IDS rules, to include these strategies. We use the policy evaluation framework to evaluate the policies and two different policy enforcement points.

The role of the security policy is to define the proper response (**correct**) of the system when a task enters the Red-Zone, based on (*i*) the system state as well as the attack severity which could be calculated from the attack metrics in conjunction with a reference to the attack (**dynamic**), (*ii*) the conditions under which a Red-Zoned task may return to its normal state (**deactivable**) and, finally, (*iii*) the response to a task breaching its ultimate limit.

Part III

The End

"If a vehicular system is not secure, it is not safe"

8 Conclusion

This thesis is concerned with the security of vehicular systems. The complexity, size and differing quality standards of the software running on the multitude of ECUs aboard a modern vehicle make the existence of security vulnerabilities a given. In addition, the lack of security mechanisms in existing vehicles enables the escalation of a compromise in one of the noncritical sub-systems (e.g., the CD player) to threaten the safety of the vehicle and its passengers. Having identified the need for a holistic solution which employs diverse security mechanisms that coexist and collaborate to guarantee the security of the system over its whole life cycle, in this thesis we propose a multilayer secure framework which aims to prevent such attacks in the first place, detect them if prevention mechanisms have failed, and finally, respond properly to any detected attacks. Applying these overlaid security protection layers decreases the chance of successful attacks and/or their consequences. In the following, we summarize the different security layers in our proposed framework, how they solve the various challenges which were introduced in chapter 1, and how they meet the main requirements stated in Section 3.1.1.

Security Analysis: The first step towards securing the in-vehicle system is by performing a security analysis to define all the existing vulnerabilities and threats as well as the agents who may exploit them. Using an individual approach which concentrates on a certain aspect (e.g., assets, attacker, or software) will not provide a complete analysis, which may in turn lead to insufficient mitigation solutions. In chapter 4, we proposed using SAVTA, a comprehensive threat model, to support security analysis for vehicular systems. SAVTA combines different existing threat modeling approaches (i.e., Software, Assets, Vulnerability, Threat, and Attacker-centric methods) to create a hybrid model. The model seeks to answer three main questions in the following ways: (1) What are we facing? By looking at the threat agents who

may threaten the vehicular system and studying their motivation, capabilities, and goals. (2) What do we have? By looking at all the assets that we have, identifying their vulnerabilities, and linking these vulnerabilities with predefined threat agents. (3) Finally, what do we need? By defining the security requirements for each of these assets to prevent potential threat agents from achieving their goals.

Secure Communication The massive number of attacks against the in-vehicle network is a result of the lack of: (1) proper authorization mechanisms which specify permissions and prohibitions to deny malicious parties from communicating with other system components. (2) data integrity and authentication mechanisms to prevent unauthorized parties from altering or sending false data. (3) And to a lesser extent, data confidentiality mechanisms which prevents unauthorized parties from disclosing the exchanged data. The reasons behind the lack of such mechanisms are numerous. In the case of authorization mechanisms, the absence of a security policy and the use of a single point (i.e., central gateway) to enforce this policy (if it exists) are the reasons behind inappropriate authorization mechanisms. The difficulty of creating a comprehensive security policy, which determines who should talk to whom and under which conditions, leads many OEMs to ignore the need for one altogether. In other cases, the late development (i.e., during the final system integration) of the security parameters may lead to incorrect access control rules. These rules may violate the security requirements specified in previous phases of the development life cycle. In Chapter 5, we proposed a methodology supporting the gradual definition of the security policy, starting from the designer of the component and then adapting and specializing the policy as the component is linked to other components and eventuality integrated with the rest of the system.

Moreover, instead of having a single place to enforce our defined security policy, we proposed a distributed access-control framework (influenced by the distributed firewall [Ioa+00]) to allow only authorized components to communicate with each other, both inside the vehicle and with external entities. Each ECU is equipped with a security module acting as a connection ingress policy checker by vetting incoming and outgoing communication and enforces the distributed security policy locally.

Regarding the adoption of security mechanisms which ensure the integrity and confidentiality of the connection, the perceived overhead of using such mechanisms has been a significant factor in the resistance to including them. In Section 5.5, we have shown that using IPsec ensured high security protection while the increase in overhead was almost negligible when using AH to safeguard (data and origin integrity). In addition, our measurements showed that even when using ESP (data confidentiality and integrity), the increased latency overhead is almost 2%.

We believe that our results dispel the myth that using secure communications protocol (such as IPsec) are too expensive for vehicular communication.

These results should encourage other researchers to adopt secure communication protocols in their system, especially if we consider the continence improvements in processing capabilities of the micro-controllers.

Intrusion Detection: Using prevention mechanisms to protect the system is not sufficient by itself. It is critical to also have a run-time mechanism to detect the occurrence of malicious activities. Anomaly-detection methods were adopted to achieve that goal. Anomaly-based mechanisms monitor the behavior of the component to identify any misuse that falls outside the predefined profile representing the component's nominal behavior. In chapter 6, we proposed using the temporal properties of safety-critical applications to construct their nominal behavior. We showed how real-time safety-driven specifications (i.e., WCRT, WCET, and deadline) cannot be used to detect malicious behavior (from the point of view of security) of the observed components. Therefore, we introduced the Red-Zone principle to impose tighter or stricter specifications to create an inner layer of defense which gives the system the ability to detect the presence of suspicious activities. We use the time window, when the task enters the Red-Zone area until reaching an ultimate boundary, to monitor the task to discover the reason behind the malicious activities, and in the case of an attack, we can use this knowledge later to fix the vulnerabilities which led to it.

Intrusion Response The security policy in many systems is defined in binary actions; the security monitor keeps the task going while it behaves normally. Otherwise, it will be terminated (losing availability). In a system where a tighter security policy is adopted, the chance of having false positive alarms is high. If the security system responds to a false alarm by terminating the suspicious components, it may lead to a continuous loss of some component availability, which may cause the vehicle to stop functioning. In chapter 7, we proposed an intrusion response framework for in-vehicle systems based on the Red-Zone principle (which was discussed in Section 6.1). Within this framework, we present different response strategies which could be applied locally (on the ECU which runs the malicious component) and globally (on the other ECUs which contain components dependent on the malicious one) before reaching the point where the attack become a failure and the task need to be terminated. Our efforts in this chapter can be considered as an introduction to implementing such a system.

8.1 Future work

We made a deliberate decision to focus on the performance overhead of our proposed security module (mainly IPsec); the reason behind this decision was that there was concern that automotive in-vehicle system hardware (i.e., ECUs) is not capable of implementing conventional approaches similar to the one we have implemented. Other evaluation measurements such as the measurement of the complexity of developing, maintaining, and managing

the security policies for the massive number of components is one relevant direction to investigate in the future.

In Chapter 6, we presented a host-based intrusion detection framework by using the temporal properties of the hard real-time components. We also introduced a network-based IDS by adopting the same methodology as for the host-based one. We left the more detailed investigation of the temporal-based networked IDS as well as the decision about when to use the temporal-based host and/or network-based IDS for future work.

Within our proposed intrusion response system (see Chapter 7), we did not explain how to calculate the cost of the response to the security incident or the cost of the possible damage. We envisage a future direction of research to use modeling technologies to estimate the cost of such response options and use that to determine the optimal response strategy.

Finally, although we have focused only on in-vehicle communication, securing the vehicle's V2X communication is achievable by adopting our framework. To achieve that, we need to implement our different security mechanisms in the On-Board Unit (OBU) which is used to support the communication between the in-vehicle components and the outside world (we already started that in our work [HAP18]). However, developing the secure communication policies for components belonging to different vehicles requires the existence of a central Vehicular Public-Key Infrastructure (VPKI). Defining and developing a cross-domain VPKI trust model and integrating it with our framework are left as future work. Furthermore, studying the effect of the intrusion response of one vehicle on the other vehicles sharing the same road needs to be researched.

A PUBLICATIONS

This chapter lists my publications. The first section includes publications directly related to the main topic of this thesis, while the second section contains other publications. Publications are ordered by the date of appearance in reverse chronological order.

A.1 Related to the Thesis

1. Mohammad Hamad, Marinos Tsantekidis, and Vassilis Prevelakis,. Red-Zone: Towards an Intrusion Response Framework for Intra-Vehicle System. In Proceedings of the 5th International Conference on Vehicle Technology and Intelligent Transport Systems (VEHITS), 2019.

2. Mohammad Hamad, Marinos Tsantekidis, and Vassilis Prevelakis,.: Intrusion Response System for Vehicles: Challenges and Vision, Smart Cities, Green Technologies, and Intelligent Transport Systems, Springer, 2019. (Submitted).

3. Mohammad Hamad, Mustafa R. Agha, and Vassilis Prevelakis, ProSEV: Proxy-Based Secure and Efficient Vehicular Communication, in 2018 IEEE Vehicular Networking Conference (VNC), (Taipei, Taiwan), IEEE, 2018.

4. Mohammad Hamad, Zain A. H. Hammadeh, Selma Saidi, Vassilis Prevelakis, and Rolf Ernst. Prediction of abnormal temporal behavior in real-time systems. In Proceedings of the 33rd Annual ACM Symposium on Applied Computing, SAC '18, pages 359-367, 2018.

5. Mohammad Hamad, Marcus Nolte, and Vassilis Prevelakis. Towards comprehensive threat modeling for vehicles. In 1st Workshop on Security and Dependability of Critical Embedded Real-Time Systems, colocated with the IEEE Real-Time Systems Symposium, pages 31-36, 2016.

6. Mohammad Hamad, Marcus Nolte, and Vassilis Prevelakis. A framework for policy based secure intra vehicle communication. In 2017 IEEE Vehicular Networking Conference (VNC), pages 1-8, Nov 2017.

7. Mohammad Hamad and Vassilis Prevelakis. Implementation and performance evaluation of embedded ipsec in microkernel os. In 2015 World Symposium on Computer Networks and Information Security (WSCNIS), pages 1-7, Sept 2015.

8. Mohammad Hamad and Vassilis Prevelakis. Secure APIs for applications in microkernel-based systems. In Proceedings of the 3rd International Conference on Information Systems Security and Privacy - Volume 1: ICISSP,, pages 553-558. INSTICC, SciTePress, 2017.

9. Mohammad Hamad, Johannes Schlatow, Vassilis Prevelakis, and Rolf Ernst. A communication framework for distributed access control in microkernel-based systems. In 12th Annual Workshop on Operating Systems Platforms for Embedded Real-Time Applications (OSPERT16), 2016.

10. Vassilis Prevelakis and Mohammad Hamad. Extending the operational envelope of applications. In 8th International Conference on Trust & Trust-worthy Computing (TRUST 2015), 2015.

11. Vassilis Prevelakis and Mohammad Hamad. A policy-based communications architecture for vehicles. In 2015 International Conference on Information Systems Security and Privacy (ICISSP), pages 155-162. IEEE, 2015.

12. Vassilis Prevelakis and Mohammad Hamad. Controlling Change via Policy Contracts. In Internet of Things Software Update Workshop (IoTSU 2016), 2016.

A.2 Others

1. Marinos Tsantekidis, Mohammad Hamad, Vassilis Prevelakis and Mustafa R. Agha. Security for Heterogeneous Systems. Heterogeneous Computing Architecture Challenges and Vision, Taylor & Francis Ltd,2019. (Book Chapter).

2. Vassilis Prevelakis, Mohammad Hamad, Jihane Najar, and Ilias Spais, Secure Data Exchange for Computationally Constrained Devices, In International workshop on Information & Operational Technology (IT & OT) security systems (IOSec 2019), Luxembourg, 2019.

3. Muhammad Ali Siddiqi, Robert M. Seepers, Mohammad Hamad, Vassilis Prevelakis, and Christos Strydis. Attack-tree-based Threat Modeling of Medical Implants. In the 7th International Workshopon Security Proofs for Embedded Systems

List of Figures

List of Tables

Abbreviations

AUTOSAR AUTomotive Open Systems ARchitecture.

CAN Controller Area Network.

E/E Electrical and Electronic.

ECU Electronic Control Unit.

GPS Global Positioning System.

IDS Intrusion Detection System.

IPC Inter Process Communication.

IPsec Internet Protocol Security.

IRS Intrusion Response System.

LIN Local Interconnect Network.

MAC Message Authentication Code.

MOST Media Oriented Systems Transport.

OEM Original Equipment Manufacturer.

OS Operating System.

OTA Over-The-Air.

RTE Run-Time Environment.

SPP Static Priority Preemptive.

V2V Vehicle-to-Vehicle.

V2X Vehicle-to-everything.

Bibliography

[Adm+16] National Highway Traffic Safety Administration et al. "Cybersecurity best practices for modern vehicles". In: *Report No. DOT HS* 812 (2016), p. 333.

[AH10] Alexandra Aguiar and Fabiano Hessel. "Embedded systemsv́irtualization: The next challenge?" In: *Proceedings of 2010 21st IEEE International Symposium on Rapid System Protyping*. IEEE. 2010, pp. 1–7.

[Ahm+09] Faraz Ahmed, Haider Hameed, M Zubair Shafiq, and Muddassar Farooq. "Using spatio-temporal information in API calls with machine learning algorithms for malware detection". In: *Proceedings of the 2nd ACM Workshop on Security and Artificial Intelligence*. ACM. 2009, pp. 55–62.

[Aij+06] Amer Aijaz, Bernd Bochow, Florian Dötzer, Andreas Festag, Matthias Gerlach, Rainer Kroh, and Tim Leinmüller. "Attacks on inter vehicle communication systems-an analysis". In: *3rd International Workshop on Intelligent Transportation (WIT 2006)*. 2006.

[AJ+19] Omar Y Al-Jarrah, Carsten Maple, Mehrdad Dianati, David Oxtoby, and Alex Mouzakitis. "Intrusion Detection Systems for Intra-Vehicle Networks: A Review". In: *IEEE Access* 7 (2019), pp. 21266–21289.

[Ala18] Swawibe Alam. "Securing vehicle Electronic Control Unit (ECU) communications and stored data". PhD thesis. 2018.

[Ale+07] Richard L Alena, John P Ossenfort, Kenneth I Laws, Andre Goforth, and Fernando Figueroa. "Communications for integrated modular avionics". In: *2007 IEEE Aerospace Conference*. IEEE. 2007, pp. 1–18.

[AM14] Hamda Hasan AlBreiki and Qusay H Mahmoud. "Evaluation of static analysis tools for software security". In: *2014 10th International Conference on Innovations in Information Technology (IIT)*. IEEE. 2014, pp. 93–98.

[AMS16] Sam Abbott-McCune and Lisa A Shay. "Intrusion prevention system of automotive network CAN bus". In: *2016 IEEE International Carnahan Conference on Security Technology (ICCST)*. IEEE. 2016, pp. 1–8.

[And72] James P Anderson. *Computer Security Technology Planning Study. Volume 2*. Tech. rep. DTIC Document, 1972.

[And+94] Debra Anderson, Thane Frivold, Ann Tamaru, and Alfonso Valdes. "Next-generation intrusion detection expert system (NIDES), software users manual, beta-update release". In: *Computer Science Laboratory, SRI International, Menlo Park, CA, USA, Technical Report SRI-CSL-95-0* (1994).

[And98] Ross Anderson. "On the security of digital tachographs". In: *European Symposium on Research in Computer Security*. Springer. 1998, pp. 111–125.

[Arn+15] Florian Arnold, Dennis Guck, Rajesh Kumar, and Mariële Stoelinga. "Sequential and parallel attack tree modelling". In: *International Conference on Computer Safety, Reliability, and Security*. Springer. 2015, pp. 291–299.

[ASM05] Brad Arkin, Scott Stender, and Gary McGraw. "Software penetration testing". In: *IEEE Security & Privacy* 3.1 (2005), pp. 84–87.

[Aud+95] Neil C Audsley, Alan Burns, Robert I Davis, Ken W Tindell, and Andy J Wellings. "Fixed priority pre-emptive scheduling: An historical perspective". In: *Real-Time Systems* 8.2-3 (1995), pp. 173–198.

[AUT14] AUTOSAR AUTOSAR. *Specification of Operating System*. Release 5.3.0. 2014.

[AUT16] AUTOSAR AUTOSAR. *AUTomotive Open System ARchitecture*. Release 4.1. 2016.

[AUT17] AUTOSAR Consortium. *Specification of Secure Onboard Communication*. Release 4.3.1. 2017.

[Avi+04] Algirdas Avizienis, J-C Laprie, Brian Randell, and Carl Landwehr. "Basic concepts and taxonomy of dependable and secure computing". In: *IEEE transactions on dependable and secure computing* 1.1 (2004), pp. 11–33.

[BAG15] Tamás Bécsi, Szilárd Aradi, and Péter Gáspár. "Security issues and vulnerabilities in connected car systems". In: *2015 International Conference on Models and Technologies for Intelligent Transportation Systems (MT-ITS)*. IEEE. 2015, pp. 477–482.

[Bal+03] Ivan Balepin, Sergei Maltsev, Jeff Rowe, and Karl Levitt. "Using specification-based intrusion detection for automated response". In: *International Workshop on Recent Advances in Intrusion Detection*. Springer. 2003, pp. 136–154.

[Bal+98] Felice Balarin, Luciano Lavagno, Praveen Murthy, Alberto Sangiovanni-Vincentelli, et al. "Scheduling for embedded real-time systems". In: *IEEE Design & Test of Computers* 15.1 (1998), pp. 71–82.

[Bay+15] Stephanie Bayer, Thomas Enderle, Dennis-Kengo Oka, and Marko Wolf. "Security crash test-practical security evaluations of automotive onboard it components". In: *Automotive-Safety & Security 2014* (2015).

[Bay+16] Stephanie Bayer, Thomas Enderle, Dennis-Kengo Oka, and Marko Wolf. "Automotive Security Testing—The Digital Crash Test". In: *Energy Consumption and Autonomous Driving*. Springer, 2016, pp. 13–22.

[BCK96] Mihir Bellare, Ran Canetti, and Hugo Krawczyk. "Keying hash functions for message authentication". In: *Annual international cryptology conference*. Springer. 1996, pp. 1–15.

[BCP02] Guillem Bernat, Antoine Colin, and Stefan M Petters. "WCET analysis of probabilistic hard real-time systems". In: *23rd IEEE Real-Time Systems Symposium, 2002. RTSS 2002.* IEEE. 2002, pp. 279–288.

[BCV01] Elmarie Biermann, Elsabe Cloete, and Lucas M Venter. "A comparison of intrusion detection systems". In: *Computers & Security* 20.8 (2001), pp. 676–683.

[BD12] Leyla Bilge and Tudor Dumitraş. "Before we knew it: an empirical study of zero-day attacks in the real world". In: *Proceedings of the 2012 ACM conference on Computer and communications security*. ACM. 2012, pp. 833–844.

[Ber14] Peter Bergmiller. "Towards Functional Safety in Drive-by-Wire Vehicles". PhD thesis. Technische Universität Braunschweig, 2014.

[Ber+17] Cinzia Bernardeschi, Marco Di Natale, Gianluca Dini, and Dario Varano. "Modeling and generation of secure component communications in AUTOSAR". In: *Proceedings of the Symposium on Applied Computing*. ACM. 2017, pp. 1473–1480.

[Ber+19] Emery D. Berger, Celeste Hollenbeck, Petr Maj, Olga Vitek, and Jan Vitek. "On the Impact of Programming Languages on Code Quality". In: *CoRR* abs/1901.10220 (2019). arXiv: 1901.10220. URL: http://arxiv.org/abs/1901.10220.

[BFL96] Matt Blaze, Joan Feigenbaum, and Jack Lacy. "Decentralized trust management". In: *Security and Privacy, 1996. Proceedings., 1996 IEEE Symposium on*. 1996.

[BFM97] Victor Braberman, Miguel Felder, and Martina Marré. "Testing Timing Behavior of Real-Time Software". In: *In International Software Quality Week*. 1997.

[BFT09] Dominique Bertrand, Sébastien Faucou, and Yvon Trinquet. "An analysis of the AUTOSAR OS timing protection mechanism". In: *Emerging Technologies & Factory Automation, 2009. ETFA 2009. IEEE Conference on*. IEEE. 2009, pp. 1–8.

[BKM07] Guillaume Bonfante, Matthieu Kaczmarek, and Jean-Yves Marion. "Control flow graphs as malware signatures". In: *International workshop on the Theory of Computer Viruses*. 2007.

[BKR00] Mihir Bellare, Joe Kilian, and Phillip Rogaway. "The security of the cipher block chaining message authentication code". In: *Journal of Computer and System Sciences* 61.3 (2000), pp. 362–399.

[Bla+99] Matt Blaze, Joan Feigenbaum, John Ioannidis, and Angelos D. Keromytis. *The KeyNote Trust-Management System Version 2*. RFC 2704. 1999.

[Bog16] Jacob Bogage. *Scary glitch affects luxury cars*. `https://www.bostonglobe.com/lifestyle/2016/06/09/scary-glitch-affects-luxury-cars/kj4wg2lhphlJDC3gATGuPM/story.html`. June 2016.

[Bos86] Robert Bosch. *Bosch automotive handbook*. 1986, pp. 1086–1087.

[Bou+12] Alexandre Bouard, Benjamin Glas, Anke Jentzsch, Alexander Kiening, Thomas Kittel, and B Weyl. "Driving automotive middleware towards a secure ip-based future". In: *Proc. of ESCAR* (2012).

[Bro06a] Manfred Broy. "Challenges in Automotive Software Engineering". In: *Proceedings of the 28th International Conference on Software Engineering*. ICSE 06. Shanghai, China: ACM, 2006, pp. 33–42. ISBN: 1-59593-375-1.

[Bro06b] Manfred Broy. "Challenges in automotive software engineering". In: *Proceedings of the 28th international conference on Software engineering*. ACM. 2006, pp. 33–42.

[Bro+07] Manfred Broy, Ingolf H Krüger, Alexander Pretschner, and Christian Salzmann. "Engineering automotive software". In: *Proceedings of the IEEE* 95.2 (2007), pp. 356–373.

[Bro+09] Manfred Broy, Mario Gleirscher, Peter Kluge, Wolfgang Krenzer, Stefano Merenda, and Doris Wild. *Automotive architecture framework: Towards a holistic and standardised system architecture description. White paper*. Tech. rep. IBM Corporation. Technical Report, Technische Universität München. TUM-I0915, 2009.

[Bur+12] Simon Burton, Jürgen Likkei, Priyamvadha Vembar, and Marko Wolf. "Automotive Functional Safety = Safety + Security". In: *Proceedings of the First International Conference on Security of Internet of Things*. SecurIT '12. Kollam, India: ACM, 2012, pp. 150–159. ISBN: 978-1-4503-1822-8. DOI: 10.1145/2490428.2490449.

[Cam+10] Mark Campbell, Magnus Egerstedt, Jonathan P How, and Richard M Murray. "Autonomous driving in urban environments: approaches, lessons and challenges". In: *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences* 368.1928 (2010), pp. 4649–4672.

[Can+99] Ran Canetti, Juan Garay, Gene Itkis, Daniele Micciancio, Moni Naor, and Benny Pinkas. "Multicast security: a taxonomy and some efficient constructions". In: *IEEE INFOCOM '99. Conference on Computer Communications. Proceedings. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. The Future is Now (Cat. No.99CH36320)*. Vol. 2. Mar. 1999, 708–716 vol.2. DOI: 10.1109/INFCOM.1999.751457.

[Car03] James V Carroll. "Vulnerability assessment of the US transportation infrastructure that relies on the global positioning system". In: *The Journal of Navigation* 56.2 (2003), pp. 185–193.

[Car+07] Richard A Caralli, James F Stevens, Lisa R Young, and William R Wilson. *Introducing octave allegro: Improving the information security risk assessment process*. Tech. rep. Carnegie-Mellon Univ Pittsburgh PA Software Engineering Inst, 2007.

[Cas07] Timothy Casey. "Threat agent library helps identify information security risks". In: *Intel White Paper* 2 (2007).

[CB04] Brian Caswell and Jay Beale. *Snort 2.1 intrusion detection*. Elsevier, 2004.

[CBR03] William R. Cheswick, Steven M. Bellovin, and Aviel D. Rubin. *Firewalls and Internet Security: Repelling the Wily Hacker*. 2nd ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2003. ISBN: 020163466X.

[CCC13] CCC. *Controlling Concurrent Change (CCC)*. 2013.

[CD16] Victor Costan and Srinivas Devadas. "Intel SGX Explained." In: *IACR Cryptology ePrint Archive* 2016.086 (2016), pp. 1–118.

[Cha09] Robert Charette. *This car runs on code*. http://www.spectrum.ieee.org/feb09/7649. Feb. 2009.

[Che+07] Xi Chen, Juejing Feng, Martin Hiller, and Vera Lauer. "Application of software watchdog as a dependability software service for automotive safety relevant systems". In: *Dependable Systems and Networks, 2007. DSN'07. 37th Annual IEEE/IFIP International Conference on*. IEEE. 2007, pp. 618–624.

[Che+11] Stephen Checkoway, Damon McCoy, Brian Kantor, Danny Anderson, Hovav Shacham, Stefan Savage, Karl Koscher, Alexei Czeskis, Franziska Roesner, Tadayoshi Kohno, et al. "Comprehensive Experimental Analyses of Automotive Attack Surfaces." In: *USENIX Security Symposium*. 2011.

[Che11] James Cheney. "A formal framework for provenance security". In: *Computer Security Foundations Symposium (CSF), 2011 IEEE 24th*. 2011.

[Chu00] Richard J. Chutorash. *Firewall for vehicle communication bus*. http://www.google.de/patents/WO2000009363A1?cl=en. WO Patent App. PCT/US1999/017,852. Feb. 2000.

[Chu+97] Yang-Hua Chu, Joan Feigenbaum, Brian LaMacchia, Paul Resnick, and Martin Strauss. "REFEREE: Trust management for Web applications". In: *Computer Networks and ISDN systems* 29.8-13 (1997), pp. 953–964.

[CM04] Brian Chess and Gary McGraw. "Static analysis for security". In: *IEEE security & privacy* 2.6 (2004), pp. 76–79.

[CM16] Riccardo Coppola and Maurizio Morisio. "Connected car: technologies, issues, future trends". In: *ACM Computing Surveys (CSUR)* 49.3 (2016), p. 46.

[CMQ87] Smoot Carl-Mitchell and John S. Quarterman. *Using ARP to implement transparent subnet gateways*. RFC 1027. `http://www.rfc-editor.org/rfc/rfc1027.txt`. Oct. 1987. URL: `http://www.rfc-editor.org/rfc/rfc1027.txt`.

[Com+14] SAE On-Road Automated Vehicle Standards Committee et al. "Taxonomy and definitions for terms related to on-road motor vehicle automated driving systems". In: *SAE Standard J* 3016 (2014), pp. 1–16.

[Con+14] PRESERVE Consortium et al. *PREparing SEcuRe VEhicle-to-X Communication Systems*. 2014. URL: `https://www.preserve-project.eu/sites/preserve-project.eu/files/PRESERVE-D1.3-V2X_Security_Architecture_V2.pdf`.

[Cum+12] Rodney Cummings, Kai Richter, Rolf Ernst, Jonas Diemer, and Arkadeb Ghosal. "Exploring use of Ethernet for in-vehicle control applications: AFDX, TTEthernet, EtherCAT, and AVB". In: *SAE International Journal of Passenger Cars-Electronic and Electrical Systems* 5.2012-01-0196 (2012), pp. 72–88.

[Cyb14] Critical Infrastructure Cybersecurity. "Framework for Improving Critical Infrastructure Cybersecurity". In: *Framework* 1 (2014), p. 11.

[Dan09] Al Danial. *CLOC–count lines of code*. `https://github.com/AlDanial/cloc`. 2009.

[DC+14] Felipe Domingos Da Cunha, Azzedine Boukerche, Leandro Villas, Aline Carneiro Viana, and Antonio AF Loureiro. "Data communication in VANETs: a survey, challenges and applications". PhD thesis. INRIA Saclay; INRIA, 2014.

[DCF07] Hervé Debar, David Curry, and Benjamin Feinstein. *The intrusion detection message exchange format (IDMEF)*. Tech. rep. 2007.

[DDW99] Hervé Debar, Marc Dacier, and Andreas Wespi. "Towards a taxonomy of intrusion-detection systems". In: *Computer Networks* 31.8 (1999), pp. 805–822.

[Deb00] Herve Debar. "An introduction to intrusion-detection systems". In: *Proceedings of Connect* 2000 (2000).

[Deb+07] Hervé Debar, Yohann Thomas, Frédéric Cuppens, and Nora Cuppens-Boulahia. "Enabling automated threat response through the use of a dynamic security policy". In: *Journal in Computer Virology* 3.3 (2007), pp. 195–210.

[Dek17] Dekra. *Accidents in Germany Caused by Driver Error*. Sept. 2017. URL: https://www.dekra-roadsafety.com/en/accidents-in-germany-caused-by-driver-error/.

[DES17] JEFF DESJARDINS. *How Many Millions of Lines of Code Does It Take?* https://www.visualcapitalist.com/millions-lines-of-code/. [Online; accessed 17-Jan-2019]. 2017.

[DNSV10] Marco Di Natale and Alberto Luigi Sangiovanni-Vincentelli. "Moving from federated to integrated architectures in automotive: The role of standards, methods and tools". In: *Proceedings of the IEEE* 98.4 (2010), pp. 603–620.

[DR01] Joan Daemen and Vincent Rijmen. "Specification for the advanced encryption standard (AES)". In: *Federal Information Processing Standards Publication* 197 (2001).

[DTB93] Robert I Davis, Ken W Tindell, and Alan Burns. "Scheduling slack time in fixed priority pre-emptive systems". In: *Real-Time Systems Symposium, 1993., Proceedings.* Dec. 1993, pp. 222–231. DOI: 10.1109/REAL.1993.393496.

[Dun02] Adam Dunkels. *lwIP-a lightweight TCP/IP stack*. 2002. (Visited on 05/19/2016).

[Dun13] Michael Dunn. "Toyota's killer firmware: Bad design and its consequences". In: *EDN Network* 28 (2013).

[DW+14] Joost CF De Winter, Riender Happee, Marieke H Martens, and Neville A Stanton. "Effects of adaptive cruise control and highly automated driving on workload and situation awareness: A review of the empirical evidence". In: *Transportation research part F: traffic psychology and behaviour* 27 (2014), pp. 196–217.

[Dwo05] Morris Dworkin. "Recommendation for Block Cipher Modes of Operation". In: *NIST Special Publication* 800 (2005), 38B.

[Ede16] Michael Eder. "Hypervisor-vs. Container-based Virtualization". In: *Network* 1 (2016).

[Edg+06] Kenneth S Edge, George C Dalton, Richard A Raines, and Robert F Mills. "Using attack and protection trees to analyze threats and defenses to homeland security". In: *Military Communications Conference, 2006. MILCOM 2006. IEEE*. IEEE. 2006, pp. 1–7.

[EES01] Jakob Engblom, Andreas Ermedahl, and Friedhelm Stappert. "A worst-case execution-time analysis tool prototype for embedded real-time systems". In: *Proc. RT-TOOLS* (2001).

[EH13] Kevin Elphinstone and Gernot Heiser. "From L3 to seL4 – What Have We Learnt in 20 Years of L4 Microkernels?" In: *ACM Symposium on Operating Systems Principles*. Farmington, PA, USA, Nov. 2013, pp. 133–150.

[Eic+95] T. von Eicken, A. Basu, V. Buch, and W. Vogels. "U-Net: A User-level Network Interface for Parallel and Distributed Computing". In: *Proceedings of the Fifteenth ACM Symposium on Operating Systems Principles (SOSP)*. Copper Mountain, Colorado, USA: ACM, 1995, pp. 40–53. ISBN: 0-89791-715-4. DOI: 10.1145/224056.224061.

[Eis+08] Thomas Eisenbarth, Timo Kasper, Amir Moradi, Christof Paar, Mahmoud Salmasizadeh, and Mohammad T Manzuri Shalmani. "On the power of power analysis in the real world: A complete break of the KeeLoq code hopping scheme". In: *Annual International Cryptology Conference*. Springer. 2008, pp. 203–220.

[EK13] Claudia Eckert and Thomas Kittel. "Security Issues of Multi-Core Architectures—The Automotive Case". In: *it-Information Technology Methoden und innovative Anwendungen der Informatik und Informationstechnik* 55.1 (2013), pp. 5–9.

[Eng17] Dag Eng. "Integrated Threat Modelling". MA thesis. 2017.

[ERS10] ERSI. *Intelligent Transport Systems (ITS); Security; Threat, Vulnerability and Risk Analysis (TVRA)*. Technical Report. ETSI, 2010.

[Eyk+17] Kevin Eykholt, Ivan Evtimov, Earlence Fernandes, Bo Li, Amir Rahmati, Chaowei Xiao, Atul Prakash, Tadayoshi Kohno, and Dawn Song. "Robust physical-world attacks on deep learning models". In: *arXiv preprint arXiv:1707.08945* (2017).

[Eyk+18] Kevin Eykholt, Ivan Evtimov, Earlence Fernandes, Bo Li, Amir Rahmati, Chaowei Xiao, Atul Prakash, Tadayoshi Kohno, and Dawn Song. "Robust physical-world attacks on deep learning visual classification". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 1625–1634.

[FAS15] Dieter Spaar Fabian A. Scherschel. *Beemer, Open Thyself! – Security vulnerabilities in BMW's ConnectedDrive*. https://www.heise.de/ct/artikel/Beemer-Open-Thyself-Security-vulnerabilities-in-BMW-s-ConnectedDrive-2540957.html. May 2015.

[FDC11] Aurélien Francillon, Boris Danev, and Srdjan Capkun. "Relay attacks on passive keyless entry and start systems in modern cars". In: *Proceedings of the Network and Distributed System Security Symposium (NDSS)*. Eidgenössische Technische Hochschule Zürich, Department of Computer Science. 2011.

[Fei+09] Martin Feilkas, Alexander Harhurin, Judith Hartmann, Daniel Ratiu, and Wolfgang Schwitzer. *Motivation and Introduction of a System of Abstraction Layers for Embedded Systems*. Tech. rep. 2009.

[FL15a] Daniel Fallstrand and Viktor Lindström. "Applicability analysis of intrusion detection and prevention in automotive systems". 53. MA thesis. 2015.

[FL15b] Daniel Fallstrand and Viktor Lindström. "Applicability analysis of intrusion detection and prevention in automotive systems". In: *Master's Thesis in Computer Systems and Networks on the Chalmers University of Technology Goteborg* (2015).

[FM91] Kevin Forsberg and Harold Mooz. "The relationship of system engineering to the project cycle". In: *INCOSE International Symposium*. Vol. 1. 1. Wiley Online Library. 1991, pp. 57–65.

[Foo+05] Bingrui Foo, Y-S Wu, Y-C Mao, Saurabh Bagchi, and Eugene Spafford. "ADEPTS: Adaptive intrusion response using attack graphs in an e-commerce environment". In: *2005 International Conference on Dependable Systems and Networks (DSN'05)*. IEEE. 2005, pp. 508–517.

[Foo+08] Bingrui Foo, Matthew W Glause, Gaspar M Howard, Yu-Sung Wu, Saurabh Bagchi, and Eugene H Spafford. "Intrusion response systems: a survey". In: *QianY. JoshiJ. TipperD. KrishnamurthyP.(Eds.), Information assurance: Dependability and security in networked systems* (2008), pp. 377–412.

[Gen19] Genode Labs GmbH. *GENODEOperating System Framework 19.05*. `https://genode.org/documentation/genode-foundations-19-05.pdf` [last access on July 2019]. 2019.

[Gla+10] Michael Glass, Daniel Herrscher, Herbert Meier, Peter Schoo, et al. "Seis security in embedded IP-based systems". In: *ATZelektronik worldwide* (2010).

[GNB15] P Giambiagi, S Nair, and D Brossard. "Abbreviated language for authorization Version 1.0". In: *OASIS eXtensible Access Control Markup Language (XACML) TC https://www. oasis-open. org/committees/download. php/55228/alfa-for-xacml-v1. 0-wd01. doc* (2015).

[GR09] Andre Groll and Christoph Ruland. "Secure and authentic communication on existing in-vehicle networks". In: *2009 IEEE Intelligent Vehicles Symposium*. IEEE. 2009, pp. 1093–1097.

[GR95] Barbara Guttman and Edward A Roback. *An introduction to computer security: the NIST handbook*. DIANE Publishing, 1995.

[GT+09] Pedro Garcia-Teodoro, Jesus Diaz-Verdejo, Gabriel Maciá-Fernández, and Enrique Vázquez. "Anomaly-based network intrusion detection: Techniques, systems and challenges". In: *computers & security* 28.1-2 (2009), pp. 18–28.

[Gu+12] Zonghua Gu, Zhu Wang, Shijian Li, and Haibin Cai. "Design and Implementation of an Automotive Telematics Gateway Based on Virtualization". In: *2012 IEEE 15th International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing Workshops*. Apr. 2012, pp. 53–58. DOI: 10.1109/ISORCW.2012.20.

[Gui+16] Gates Guilbert, Ewing Jack, Russell Karl, and Watkins Deerek. *Explaining Volkswagen's Emissions Scandal*. http://www.nytimes.com/interactive/2015/business/international/vw-diesel-emissions-scandal-explained.html. July 2016.

[Had+12] Dina Hadžiosmanović, Lorenzo Simionato, Damiano Bolzoni, Emmanuele Zambon, and Sandro Etalle. "N-gram against the machine: On the feasibility of the n-gram network analysis for binary protocols". In: *International Workshop on Recent Advances in Intrusion Detection*. Springer. 2012, pp. 354–373.

[Ham+08] Mark D Hamilton, Michael Tunstall, Emanuel M Popovici, and William P Marnane. "Side channel analysis of an automotive microprocessor". In: *16th IET Irish Signals and Systems Conference*. IET, 2008.

[Ham+16] Mohammad Hamad, Johannes Schlatow, Vassilis Prevelakis, and Rolf Ernst. "A communication framework for distributed access control in microkernel-based systems". In: *12th Annual Workshop on Operating Systems Platforms for Embedded Real-Time Applications (OSPERT16)*. 2016.

[Ham+17] Zain A. H. Hammadeh, Sophie Quinton, Marco Panunzio, Rafik Henia, Laurent Rioux, and Rolf Ernst. "Budgeting Underspecified Tasks for Weakly-Hard Real-Time Systems". In: *The 29th Euromicro Conference on Real-Time Systems (ECRTS17)*. Dubrovnik, Croatia, June 2017.

[Ham+18] Mohammad Hamad, Zain AH Hammadeh, Selma Saidi, Vassilis Prevelakis, and Rolf Ernst. "Prediction of abnormal temporal behavior in real-time systems". In: *Proceedings of the 33rd Annual ACM Symposium on Applied Computing*. ACM. 2018, pp. 359–367.

[Han+04] Hans Hansson, Mikael Åkerholm, Ivica Crnkovic, and Martin Torngren. "SaveCCM-a component model for safety-critical real-time systems". In: *Proceedings. 30th Euromicro Conference, 2004.* IEEE. 2004, pp. 627–635.

[Han+14] Gang Han, Haibo Zeng, Yaping Li, and Wenhua Dou. "SAFE: Security-aware flexray scheduling engine". In: *Proceedings of the conference on Design, Automation & Test in Europe*. European Design and Automation Association. 2014, p. 8.

[Han+14] S. Han, M. Xie, H. Chen, and Y. Ling. "Intrusion Detection in Cyber-Physical Systems: Techniques and Challenges". In: *IEEE Systems Journal* 8.4 (Dec. 2014), pp. 1052–1062. ISSN: 1932-8184. DOI: 10.1109/JSYST.2013.2257594.

[HAP18] Mohammad Hamad, Mustafa R Agha, and Vassilis Prevelakis. "ProSEV: Proxy-Based Secure and Efficient Vehicular Communication". In: *2018 IEEE Vehicular Networking Conference (VNC)*. IEEE. 2018, pp. 1–8.

[Has+16a] Monowar Hasan, Sibin Mohan, Rakesh B Bobba, and Rodolfo Pellizzoni. "A server model to integrate security tasks into fixed-priority real-time systems". In: *2016 IEEE CERTS*. 2016.

[Has+16b] Monowar Hasan, Sibin Mohan, Rakesh B Bobba, and Rodolfo Pellizzoni. "Exploring Opportunistic Execution for Integrating Security into Legacy Hard Real-Time Systems". In: *2016 IEEE Real-Time Systems Symposium (RTSS)*. 2016, pp. 123–134. DOI: 10.1109/RTSS.2016.021.

[Hei08] Gernot Heiser. "The role of virtualization in embedded systems". In: *Proceedings of the 1st workshop on Isolation and integration in embedded systems*. ACM. 2008, pp. 11–16.

[Hen+09] Olaf Henniger, Ludovic Apvrille, Andreas Fuchs, Yves Roudier, Alastair Ruddle, and Benjamin Weyl. "Security requirements for automotive on-board networks". In: *2009 9th International Conference on Intelligent Transport Systems Telecommunications (ITST)*. IEEE. 2009.

[Her17] Nadine Herold. "Incident Handling Systems with Automated Intrusion Response". Dissertation. München: Technische Universität München, 2017.

[HF04] Reinhold Heckmann and Christian Ferdinand. "Worst-case execution time prediction by static program analysis". In: *18th International Parallel and Distributed Processing Symposium (IPDPS 2004)*. 2004.

[HKD08] Tobias Hoppe, Stefan Kiltz, and Jana Dittmann. "Adaptive dynamic reaction to automotive IT security incidents using multimedia car environment". In: *Information Assurance and Security, 2008. ISIAS'08. Fourth International Conference on*. IEEE. 2008, pp. 295–298.

[HKD09] Tobias Hoppe, Stefan Kiltz, and Jana Dittmann. "Applying intrusion detection to automotive IT-early insights and remaining challenges". In: *Journal of Information Assurance and Security (JIAS)* 4.6 (2009), pp. 226–235.

[HKD11] Tobias Hoppe, Stefan Kiltz, and Jana Dittmann. "Security threats to automotive CAN networks—Practical examples and selected short-term countermeasures". In: *Reliability Engineering & System Safety* 96.1 (2011), pp. 11–25.

[HL18] Qiang Hu and Feng Luo. "Review of Secure Communication Approaches for In-Vehicle Network". In: *International Journal of Automotive Technology* 19.5 (Oct. 2018), pp. 879–894.

[HNP16] Mohammad Hamad, Marcus Nolte, and Vassilis Prevelakis. "Towards comprehensive threat modeling for vehicles". In: *1st Workshop on Security and Dependability of Critical Embedded Real-Time Systems*. 2016, pp. 31–36.

[HNP17] Mohammad Hamad, Marcus Nolte, and Vassilis Prevelakis. "A framework for policy based secure intra vehicle communication". In: *Vehicular Networking Conference (VNC), 2017 IEEE*. IEEE. 2017.

[Hol+] Sönke Holthusen, Sophie Quinton, Ina Schaefer, Johannes Schlatow, and Martin Wegner. "Using Multi-Viewpoint Contracts for Negotiation of Embedded Software Updates". In: *EPTCS 208* (), p. 31.

[Hou+02] Russ Housley, Tim Polk, Dr. Warwick S. Ford, and David Solo. *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*. RFC 3280. 2002.

[HP15] Mohammad Hamad and Vassilis Prevelakis. "Implementation and performance evaluation of embedded IPsec in microkernel OS". In: *Computer Networks and Information Security (WSCNIS), 2015 World Symposium on*. 2015.

[HP17] Mohammad Hamad and Vassilis Prevelakis. "Secure APIs for Applications in Microkernel-based Systems". In: *Proceedings of the 3rd International Conference on Information Systems Security and Privacy - Volume 1: ICISSP,* INSTICC. SciTePress, 2017, pp. 553–558. ISBN: 978-989-758-209-7. DOI: 10.5220/0006265805530558.

[HR10] Edward S Hanawalt and William B Rouse. "Car wars: Factors underlying the success or failure of new car programs". In: *Systems Engineering* 13.4 (2010), pp. 389–404.

[HR95] Moncef Hamdaoui and Parameswaran Ramanathan. "A Dynamic Priority Assignement Technique for Streams with (m, k)-Firm Deadlines". In: *IEEE Trans. Computers* 44.12 (1995), pp. 1443–1451.

[HRS09] Olaf Henniger, Alastair Ruddle, and Hervé Seudié. "Securing vehicular on-board it systems: The evita project". In: *VDI/VW Automotive Security Conference*. 2009.

[HSM12] Peter Hank, Thomas Suermann, and Steffen Mueller. "Automotive Ethernet, a holistic approach for a next generation in-vehicle networking standard". In: *Advanced Microsystems for Automotive Applications 2012*. Springer, 2012, pp. 79–89.

[HTP19a] Mohammad Hamad, Marinos Tsantekidis, and Vassilis Prevelakis. "Intrusion Response System for Vehicles: Challenges and Visions". In: *Smart Cities, Green Technologies, and Intelligent Transport Systems*. submitted. Springer, 2019.

[HTP19b] Mohammad Hamad, Marinos Tsantekidis, and Vassilis Prevelakis. "Red-Zone: Towards an Intrusion Response Framework for Intra-Vehicle System". In: *the 5th International Conference on Vehicle Technology and Intelligent Transport Systems (VEHITS)*. Crete, Greece, May 2019.

[HZ+18] Jerry den Hartog, Nicola Zannone, et al. "Security and privacy for innovative automotive applications: A survey". In: *Computer Communications* 132 (2018), pp. 17–41.

[Idr+11] Muhammad Sabir Idrees, Hendrik Schweppe, Yves Roudier, Marko Wolf, Dirk Scheuermann, and Olaf Henniger. "Secure Automotive On-Board Protocols: A Case of Over-the-Air Firmware Updates". In: *Nets4Cars/Nets4Trains*. 2011.

[Ina+16] Zakira Inayat, Abdullah Gani, Nor Badrul Anuar, Muhammad Khurram Khan, and Shahid Anwar. "Intrusion response systems: Foundations, design, and challenges". In: *Journal of Network and Computer Applications* 62 (2016), pp. 53–74.

[Inc11] RTCA Inc. *RTCA DO-178C:Software Considerations in Airborne Systems and Equipment Certification*. ISO RTCA DO-178C. 2011.

[Ioa+00] Sotiris Ioannidis, Angelos D Keromytis, Steve M Bellovin, and Jonathan M Smith. "Implementing a distributed firewall". In: *Proceedings of the 7th ACM conference on Computer and communications security*. 2000.

[Ioa05] Sotiris Ioannidis. "Security Policy Consistency and Distributed Evaluation in Heterogeneous Environments". PhD thesis. University of Pennsylvania, 2005.

[IR12] Muhammad Sabir Idrees and Yves Roudier. "Effective and efficient security policy engines for automotive on-board networks". In: *International Workshop on Communication Technologies for Vehicles*. Springer. 2012, pp. 14–26.

[Isoa] *Information technology – Security techniques – Evaluation criteria for IT security*. Tech. rep. International Organization for Standardization, 2009.

[Isob] *Information technology — Security techniques — Information security management systems — Overview and vocabulary*. Standard. Geneva, CH: International Organization for Standardization, Jan. 2014.

[Isoc] *Information technology–Security techniques–Methodology for IT security evaluation*. Standard. International Standard ISO/IEC 18045, 2000.

[Isod] *Road vehicles – Diagnostic communication over Internet Protocol (DoIP) – Part 1: General information and use case definition*. Standard. International Organization for Standardization, 2011.

[Isoe] *Road vehicles – Local Interconnect Network (LIN) – Part 3: Protocol specification*. Standard. International Organization for Standardization, Aug. 2016.

[Isof] *Road vehicles – Open interface for embedded automotive applications – Part 1-5*. Standard. Geneva, CH: International Organization for Standardization, Feb. 2005.

[Isog] *Road vehicles - FlexRay communications system*. Standard. Geneva, CH: International Organization for Standardization, Feb. 2013.

[Isoh] *Road vehicles – Controller area network (CAN) – Part 1: Data link layer and physical*. Standard. International Organization for Standardization, Aug. 2003.

[Isoi] *Road vehicles – Functional safety*. Standard. Geneva, CH: International Organization for Standardization, Apr. 2011.

[IVS09] Michael G Iatrou, Artemios G Voyiatzis, and Dimitrios N Serpanos. "Network Stack Optimization for Improved IPsec Performance on Linux." In: *SECRYPT*. 2009, pp. 83–91.

[Izo+16] Viacheslav Izosimov, Alexandros Asvestopoulos, Oscar Blomkvist, and Martin Törngren. "Security-aware development of cyber-physical systems illustrated with automotive case study". In: *2016 Design, Automation & Test in Europe Conference & Exhibition, DATE 2016, Dresden, Germany*. 2016.

[JKB15] Lalit Mohan Joshi, Mukesh Kumar, and Rajendra Bharti. "Understanding threats in hypervisor, its forensics mechanism and its research challenges". In: *International Journal of Computer Applications* 119.1 (2015).

[JMD04] Grant A Jacoby, Randy Marchany, and NathanielJ Davis. "Battery-based intrusion detection a first line of defense". In: *Proceedings from the Fifth Annual IEEE SMC Information Assurance Workshop, 2004.* IEEE. 2004, pp. 272–279.

[Joh+13] Brittany Johnson, Yoonki Song, Emerson Murphy-Hill, and Robert Bowdidge. "Why don't software developers use static analysis tools to find bugs?" In: *Proceedings of the 2013 International Conference on Software Engineering*. IEEE Press. 2013, pp. 672–681.

[Jon+96] Rick Jones et al. "NetPerf: a network performance benchmark". In: *Information Networks Division, Hewlett-Packard Company* (1996).

[Kan+13] Wael Kanoun, Layal Samarji, Nora Cuppens-Boulahia, Samuel Dubus, and Frédéric Cuppens. "Towards a Temporal Response Taxonomy". In: *Data Privacy Management and Autonomous Spontaneous Security*. Ed. by Roberto Di Pietro, Javier Herranz, Ernesto Damiani, and Radu State. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 318–331.

[Kar+16] Trishank Karthik, Akan Brown, Sebastien Awwad, Damon McCoy, Russ Bielawski, Cameron Mott, Sam Lauzon, André Weimerskirch, and Justin Cappos. "Uptane: Securing software updates for automobiles". In: *International Conference on Embedded Security in Car*. 2016.

[KCB97] Hugo Krawczyk, Ran Canetti, and Mihir Bellare. *HMAC: Keyed-Hashing for Message Authentication*. `https://rfc-editor.org/rfc/rfc2104.txt`. Feb. 1997. DOI: `10.17487/RFC2104`.

[Ken05a] Stephen Kent. *IP Authentication Header*. `https://rfc-editor.org/rfc/rfc4302.txt`. RFC. Dec. 2005. DOI: `10.17487/RFC4302`.

[Ken05b] Stephen Kent. *IP Encapsulating Security Payload (ESP)*. `https://rfc-editor.org/rfc/rfc4303.txt`. RFC. Dec. 2005. DOI: `10.17487/RFC4303`.

[Ker12] Andreas Kern. "Ethernet and IP for Automotive E/E-Architectures-Technology Analysis, Migration Concepts and Infrastructure". PhD thesis. Erlangen, Friedrich-Alexander-Universität Erlangen-Nürnberg, 2012.

[KG99] Loren Kohnfelder and Praerit Garg. The Threat to our Products. `https://www.skyboxsecurity.com/sites/default/files/Threat-Centric_Vulnerability_Management-Solution_Brief.pdf` [last access on Feb 2019]. Apr. 1999.

[KKA17] Adi Karahasanovic, Pierre Kleberger, and Magnus Almgren. "Adapting Threat Modeling Methods for the Automotive Industry". In: *Proceedings of the 15th ESCAR Conference*. 2017, pp. 1–10.

[Koc96] Paul C Kocher. "Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems". In: *Annual International Cryptology Conference*. Springer. 1996, pp. 104–113.

[KOJ11] Pierre Kleberger, Tomas Olovsson, and Erland Jonsson. "Security aspects of the in-vehicle network in the connected car". In: *Intelligent Vehicles Symposium (IV), 2011 IEEE*. IEEE. 2011, pp. 528–533.

[Kos+10] Karl Koscher, Alexei Czeskis, Franziska Roesner, Shwetak Patel, Tadayoshi Kohno, Stephen Checkoway, Damon Mccoy, Brian Kantor, Danny Anderson, Hovav Shacham, and Stefan Savage. "Experimental security analysis of a modern automobile". In: *In Proceedings of IEEE Symposium on Security and Privacy in*. 2010.

[Koz05] Charles M Kozierok. *The TCP/IP guide: a comprehensive, illustrated Internet protocols reference*. No Starch Press, 2005.

[KP07] Angelos D Keromytis and Vassilis Prevelakis. "Designing firewalls: A survey". In: *Network Security: Current Status and Future Directions* (2007), pp. 33–49.

[KV02] Richard A Kemmerer and Giovanni Vigna. "Intrusion detection: a brief history and overview". In: *Computer* 35.4 (2002), supl27–supl30.

[Laa+09] Youssef Laarouchi, Yves Deswarte, David Powell, Jean Arlat, and Eric De Nadai. "Ensuring safety and security for avionics: A case study". In: *DAta Systems In Aerospace (DASIA 2009)* (2009), pp. 26–29.

[Lab18] Tencent Keen Security Lab. *Experimental Security Assessment of BMW Cars: A Summary Report*. `https://keenlab.tencent.com/en/Experimental_Security_Assessment_of_BMW_Cars_by_KeenLab.pdf/` [last access on March 2019]. May 2018.

[Lei+06] Tim Leinmueller, Levente Buttyan, Jean-Pierre Hubaux, Frank Kargl, Rainer Kroh, Panagiotis Papadimitratos, Maxim Raya, and Elmar Schoch. "SEVECOM - Secure Vehicle Communication". In: *IST Mobile and Wireless Communication Summit*. POST_TALK. 2006.

[Lew+01] Scott M Lewandowski, Daniel J Van Hook, Gerald C O'Leary, Joshua W Haines, and Lee M Rossey. "SARA: Survivable autonomic response architecture". In: *Proceedings DARPA Information Survivability Conference and Exposition II. DISCEX'01*. Vol. 1. IEEE. 2001, pp. 77–88.

[LF12] Congli Ling and Dongqin Feng. "An algorithm for detection of malicious messages on CAN buses". In: *2012 National Conference on Information Technology and Computer Science*. Atlantis Press. 2012.

[LH15] Jan Lastinec and Ladislav Hudec. "A performance analysis of IPSec/AH protocol for automotive environment". In: *Proceedings of the 16th International Conference on Computer Systems and Technologies*. ACM. 2015, pp. 299–304.

[Lia+15] Wenshuang Liang, Zhuorong Li, Hongyang Zhang, Shenling Wang, and Rongfang Bie. "Vehicular ad hoc networks: architectures, research issues, methodologies, challenges, and trends". In: *International Journal of Distributed Sensor Networks* 11.8 (2015), p. 745303.

[Lie93] Jochen Liedtke. "Improving IPC by kernel design". In: *ACM SIGOPS Operating Systems Review* 27.5 (Dec. 1993), pp. 175–188. DOI: `10.1145/173668.168633`.

[Lim+11] Hyung-Taek Lim, Benjamin Krebs, Lars Volker, and Peter Zahrer. "Performance evaluation of the inter-domain communication in a switched Ethernet based in-car network". In: *2011 IEEE 36th Conference on Local Computer Networks*. IEEE. 2011, pp. 101–108.

[Lin+13] Chung-Wei Lin, Qi Zhu, Calvin Phung, and Alberto Sangiovanni-Vincentelli. In: *2013 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. Nov. 2013, pp. 115–121. DOI: `10.1109/ICCAD.2013.6691106`.

[Lin15] Rainer Link. *Is Your Car Broadcasting Too Much Information?* `https://blog.trendmicro.com/trendlabs-security-intelligence/is-your-car-broadcasting-too-much-information/`. June 2015.

[LJK17] Hyunsung Lee, Seong Hoon Jeong, and Huy Kang Kim. "OTIDS: A novel intrusion detection system for in-vehicle network by using remote frame". In: *2017 15th Annual Conference on Privacy, Security and Trust (PST)*. IEEE. 2017, pp. 57–5709.

[LL96] Gérard Le Lann. "The ariane 5 flight 501 failure-a case study in system engineering for computing systems". PhD thesis. INRIA, 1996.

[LNJ08] Ulf E Larson, Dennis K Nilsson, and Erland Jonsson. "An approach to specification-based attack detection for in-vehicle networks". In: *Intelligent Vehicles Symposium, 2008 IEEE*. IEEE. 2008, pp. 220–225.

[Loc+05] Michael E Locasto, Ke Wang, Angelos D Keromytis, and Salvatore J Stolfo. "Flips: Hybrid adaptive intrusion prevention". In: *International Workshop on Recent Advances in Intrusion Detection*. Springer. 2005, pp. 82–101.

[Lod16] David Lodge. *Hacking the Mitsubishi Outlander PHEV hybrid*. `https://www.pentestpartners.com/security-blog/hacking-the-mitsubishi-outlander-phev-hybrid-suv/`. June 2016.

[Lou+19] George Loukas, Eirini Karapistoli, Emmanouil Panaousis, Panagiotis Sarigiannidis, Anatolij Bezemskij, and Tuan Vuong. "A taxonomy and survey of cyber-physical intrusion detection approaches for vehicles". In: *Ad Hoc Networks* 84 (2019), pp. 124–147.

[LS+11] Edward A Lee, Sanjit A Seshia, et al. "Introduction to embedded systems". In: *A cyber-physical systems approach* 499 (2011).

[LSL15] Sixing Lu, Minjun Seo, and Roman Lysecky. "Timing-based anomaly detection in embedded systems". In: *Design Automation Conference (ASP-DAC), 2015 20th Asia and South Pacific*. IEEE. 2015, pp. 809–814.

[LW09] Adam Lackorzynski and Alexander Warg. "Taming Subsystems: Capabilities As Universal Resource Access Control in L4". In: *Proceedings of the Second Workshop on Isolation and Integration in Embedded Systems (IIES)*. New York, NY, USA: ACM, 2009, pp. 25–30. DOI: `10.1145/1519130.1519135`.

[MA05] Arezou Mohammadi and Selim G Akl. "Scheduling algorithms for real-time systems". In: *School of Computing Queens University, Tech. Rep* (2005).

[Mar13] Edward J. Markey. *As Wireless Technology Becomes Standard, Markey Queries Car Companies about Security, Privacy*. https://www.markey.senate.gov/news/press-releases/as-wireless-technology-becomes-standard-markey-queries-car-companies-about-security-privacy. Dec. 2013.

[Mar16] IHS Markit. *Vehicles Getting Older: Average Age of Light Cars and Trucks in U.S. Rises Again in 2016 to 11.6 Years, IHS Markit Says*. https://news.ihsmarkit.com/press-release/automotive/vehicles-getting-older-average-age-light-cars-and-trucks-us-rises-again-201. Nov. 2016.

[Mas+16] Philip Masek, Magnus Thulin, Hugo Andrade, Christian Berger, and Ola Benderius. "Systematic evaluation of sandboxed software deployment for real-time software on the example of a self-driving heavy vehicle". In: *Intelligent Transportation Systems (ITSC), 2016 IEEE 19th International Conference on*. IEEE. 2016, pp. 2398–2403.

[MC14] Robert Mitchell and Ing-Ray Chen. "A survey of intrusion detection techniques for cyber-physical systems". In: *ACM Computing Surveys (CSUR)* 46.4 (2014), p. 55.

[ME15] Mischa Moestl and Rolf Ernst. "Cross-layer dependency analysis for safety-critical systems design". In: *ARCS 2015-The 28th International Conference on Architecture of Computing Systems. Proceedings*. VDE. 2015, pp. 1–7.

[Mea+18] Nancy R Mead, Forrest Shull, Krishnamurthy Vemuru, and Ole Villadsen. "A Hybrid Threat Modeling Method". In: *Carnegie MellonUniversity-Software Engineering Institute-Technical Report-CMU/SEI-2018-TN-002* (2018).

[MEL01] Andrew Moore, Robert Ellison, and Richard Linger. *Attack Modeling for Information Security and Survivability*. Tech. rep. CMU/SEI-2001-TN-001. Software Engineering Institute, Carnegie Mellon University, 2001.

[Mer97] Robyn Meredith. *VW agrees to pay G.M. $100 million in espionage suit*. https://www.nytimes.com/1997/01/10/business/vw-agrees-to-pay-gm-100-million-in-espionage-suit.html. Jan. 1997.

[MGF10] Michael Müter, André Groll, and Felix C Freiling. "A structured approach to anomaly detection for in-vehicle networks". In: *2010 Sixth International Conference on Information Assurance and Security*. IEEE. 2010, pp. 92–98.

[MHC14] Charlie McCarthy, Kevin Harnett, and Art Carter. *Characterization of potential security threats in modern automobiles: A composite modeling approach*. Tech. rep. 2014.

[MKK15] Roberto Morabito, Jimmy Kjällman, and Miika Komu. "Hypervisors vs. lightweight virtualization: a performance comparison". In: *Cloud Engineering (IC2E), 2015 IEEE International Conference on*. IEEE. 2015, pp. 386–393.

[MLY05] Suvda Myagmar, Adam J Lee, and William Yurcik. "Threat modeling as a basis for security requirements". In: Citeseer. 2005.

[Mon+18] Jean-Philippe Monteuuis, Jonathan Petit, Jun Zhang, Houda Labiod, Stefano Mafrica, and Alain Servel. "Attacker model for Connected and Automated Vehicles". In: *ACM COMPUTER SCIENCE IN CARS SYMPOSIUM (CSCS 2018)*. 2018.

[Mor+17] Roberto Morabito, Riccardo Petrolo, Valeria Loscri, Nathalie Mitton, Giuseppe Ruggeri, and Antonella Molinaro. "Lightweight virtualization as enabling technology for future smart cars". In: *Integrated Network and Service Management (IM), 2017 IFIP/IEEE Symposium on*. IEEE. 2017, pp. 1238–1245.

[Mou+16] Ahmed Refaat Mousa, Pakinam NourElDeen, Marianne Azer, and Mahmoud Allam. "Lightweight authentication protocol deployment over FlexRay". In: *Proceedings of the 10th International Conference on Informatics and Systems*. ACM. 2016, pp. 233–239.

[MP16] Sandeep Ankush Maske and Thaksen J Parvat. "Advanced anomaly intrusion detection technique for host based system using system call patterns". In: *2016 International Conference on Inventive Computation Technologies (ICICT)*. Vol. 2. IEEE. 2016, pp. 1–4.

[MTK19] Dimitris Mbakoyiannis, Othon Tomoutzoglou, and George Kornaros. "Secure Over-the-air Firmware Updating for Automotive Electronic Control Units". In: *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*. SAC '19. Limassol, Cyprus: ACM, 2019, pp. 174–181. ISBN: 978-1-4503-5933-7. DOI: 10.1145/3297280.3297299.

[Mun+15] Philipp Mundhenk, Sebastian Steinhorst, Martin Lukasiewycz, Suhaib A Fahmy, and Samarjit Chakraborty. "Lightweight authentication for secure automotive networks". In: *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*. EDA Consortium. 2015, pp. 285–288.

[MV14] Charlie Miller and Chris Valasek. "A survey of remote automotive attack surfaces". In: *Black Hat*. 2014.

[MV15] Charlie Miller and Chris Valasek. "Remote exploitation of an unaltered passenger vehicle". In: *Black Hat USA* 2015 (2015), p. 91.

[Nai+17] Alena Naiakshina, Anastasia Danilova, Christian Tiefenau, Marco Herzog, Sergej Dechand, and Matthew Smith. "Why do developers get password storage wrong?: A qualitative usability study". In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM. 2017, pp. 311–328.

[Nas+19] Dudi Nassi, Raz Ben-Netanel, Yuval Elovici, and Ben Nassi. "MobilBye: Attacking ADAS with Camera Spoofing". In: *arXiv preprint arXiv:1906.09765* (2019).

[NC03] Schild Niklaus and Scheurer Christian. "Embedded IPSec, light weight IPSec Implementation". Diploma thesis. Berne Univ, Switzerland, 2003.

[NH14] Adnan Nadeem and Michael P Howarth. "An intrusion detection & adaptive response mechanism for MANETs". In: *Ad Hoc Networks* 13 (2014), pp. 368–380.

[Nig+12] Tyler Nighswander, Brent Ledvina, Jonathan Diamond, Robert Brumley, and David Brumley. "GPS software attacks". In: *Proceedings of the 2012 ACM conference on Computer and communications security*. ACM. 2012, pp. 450–461.

[Nil+09] Dennis K Nilsson, Ulf E Larson, Francesco Picasso, and Erland Jonsson. "A first simulation of attacks in the automotive network communications protocol FlexRay". In: *Proceedings of the International Workshop on Computational Intelligence in Security for Information Systems CISIS'08*. Springer. 2009, pp. 84–91.

[Nim13] Kurt Nimmo. *Richard Clarke: Hastings Accident "Consistent with a Car Cyber Attack"*. `http://www.informationliberation.com/?id=44269` [last access on March 2019]. June 2013.

[NL08] Dennis K Nilsson and Ulf E Larson. "Secure firmware updates over the air in intelligent vehicles". In: *ICC Workshops-2008 IEEE International Conference on Communications Workshops*. IEEE. 2008, pp. 380–384.

[NL09] Dennis K Nilsson and Ulf E Larson. "A Defense-in-Depth Approach to Securing the Wireless Vehicle Infrastructure". In: *JOURNAL OF NETWORKS* 4.7 (2009), p. 553.

[NLC14] NLC. *Successful connection on the Model S internal Ethernet network*. `https://teslamotorsclub.com/tmc/threads/successful-connection-on-the-model-s-internal-ethernet-network.28185/`. Apr. 2014.

[Nol06] Thomas Nolte. "Share-driven scheduling of embedded networks". PhD thesis. Institutionen för Datavetenskap och Elektronik, 2006.

[NPR18] Vivek Nigam, Alexander Pretschner, and Harald Ruess. "Model-Based Safety and Security Engineering". In: *arXiv preprint arXiv:1810.04866* (2018).

[NSL08] Nicolas Navet and Françoise Simonot-Lion. "A review of embedded automotive protocols". In: *Automotive Embedded Systems Handbook, Industrial Information Technology Series, pages* (2008), pp. 4–1.

[NSL13] Nicolas Navet and Françoise Simonot-Lion. *In-vehicle communication networks-a historical perspective and review*. Tech. rep. University of Luxembourg, 2013.

[NXP18] NXP Semiconductor. *Automotive Gateway: A Key Component to Securing the Connected Car*. `https://www.nxp.com/docs/en/white-paper/AUTOGWDEVWPUS.pdf` [last access on March 2019]. 2018.

[Okl] *Cog Systems: OKL4 Microvisor*. `http://cog.systems/products/okl4-microvisor.shtml`[last access on Jan 2019].

[Olo14] Jennie Olofsson. "'Zombies ahead!'A study of how hacked digital road signs destabilize the physical space of roadways". In: *Visual Communication* 13.1 (2014), pp. 75–93.

[PDB14] Constantinos Patsakis, Kleanthis Dellios, and Mélanie Bouroche. "Towards a distributed secure in-vehicle communication architecture for modern vehicles". In: *Computers & Security* 40 (2014), pp. 60–74.

[Pen+17] Scott Drew Pendleton, Hans Andersen, Xinxin Du, Xiaotong Shen, Malika Meghjani, You Hong Eng, Daniela Rus, and Marcelo H Ang. "Perception, planning, control, and coordination for autonomous vehicles". In: *Machines* 5.1 (2017), p. 6.

[Pet+15] Jonathan Petit, Bas Stottelaar, Michael Feiri, and Frank Kargl. "Remote Attacks on Automated Vehicles Sensors: Experiments on Camera and LiDAR". In: *Black Hat Europe*. Nov. 2015.

[PH15a] Vassilis Prevelakis and Mohammad Hamad. "A policy-based communications architecture for vehicles". In: *2015 International Conference on Information Systems Security and Privacy (ICISSP)*. IEEE. 2015, pp. 155–162.

[PH15b] Vassilis Prevelakis and Mohammad Hamad. "Extending the Operational Envelope of Applications". In: *8th International Conference on Trust & Trustworthy Computing (TRUST 2015)*. 2015.

[Pou10] Kevin Poulsen. *Hacker Disables More Than 100 Cars Remotely*. `https://www.wired.com/2010/03/hacker-bricks-cars/`. Mar. 2010.

[PP08] Krutartha Patel and Sri Parameswaran. "SHIELD: a software hardware design methodology for security and reliability of MPSoCs". In: *Design Automation Conference, 2008. DAC 2008. 45th ACM/IEEE*. IEEE. 2008, pp. 858–861.

[Pro03] Niels Provos. "A Virtual Honeypot Framework". In: *Ann Arbor* 1001 (2003), pp. 48103–4943.

[PUB77] NIST FIPS PUB. "46-3. Data Encryption Standard". In: *Federal Information Processing Standards, National Bureau of Standards, US Department of Commerce* (1977).

[PWW08] Jan Pelzl, Marko Wolf, and Thomas Wollinger. *Virtualization technologies for cars*. Tech. rep. 2008.

[QHE12] Sophie Quinton, Matthias Hanke, and Rolf Ernst. "Formal analysis of sporadic overload in real-time systems". In: *2012 Design, Automation & Test in Europe Conference & Exhibition, DATE 2012, Dresden, Germany, March 12-16, 2012*. 2012, pp. 515–520.

[Qnx] *QNX Neutrino RTOS*. `http://www.qnx.com/products/neutrino-rtos/neutrino-rtos.html` [last access on Jan 2019].

[Qui+14] Sophie Quinton, Torsten T Bone, Julien Hennig, Moritz Neukirchner, Mircea Negrean, and Rolf Ernst. "Typical worst case response-time analysis and its use in automotive network design". In: *Proceedings of the 51st Annual Design Automation Conference*. ACM. 2014, pp. 1–6.

[Ray+14] Baishakhi Ray, Daryl Posnett, Vladimir Filkov, and Premkumar Devanbu. "A large scale study of programming languages and code quality in github". In: *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*. ACM. 2014, pp. 155–165.

[Res+14] Andreas Reschka, Marcus Nolte, Torben Stolte, Johannes Schlatow, Rolf Ernst, and Markus Maurer. "Specifying a middleware for distributed embedded vehicle control systems". In: *Proceedings of the 2014 IEEE International Conference on Vehicular Electronics and Safety (ICVES)*. Hyderabad, India, Dec. 2014, pp. 117–122. URL: `http://dx.doi.org/10.1109/ICVES.2014.7063734`.

[RMM15] J. Rieken, R. Matthaei, and M. Maurer. "Toward Perception-Driven Urban Environment Modeling for Automated Road Vehicles". In: *2015 IEEE 18th International Conference on Intelligent Transportation Systems*. 2015. DOI: `10.1109/ITSC.2015.124`.

[Ros09] Matthew Rosenquist. "Prioritizing Information Security Risks with Threat Agent Risk Assessment". In: *Intel Corporation White Paper* (2009).

[Rou+10] Ishtiaq Rouf, Rob Miller, Hossen Mustafa, Travis Taylor, Sangho Oh, Wenyuan Xu, Marco Gruteser, Wade Trappe, and Ivan Seskar. "Security and Privacy Vulnerabilities of In-car Wireless Networks: A Tire Pressure Monitoring System Case Study". In: *Proceedings of the 19th USENIX Conference on Security*. USENIX Security'10. Berkeley, CA, USA: USENIX Association, 2010. ISBN: 888-7-6666-5555-4.

[RSA78] Ronald L Rivest, Adi Shamir, and Leonard Adleman. "A method for obtaining digital signatures and public-key cryptosystems". In: *Communications of the ACM* 21.2 (1978), pp. 120–126.

[Rum+19] M. Rumez, A. Duda, P. Gründer, R. Kriesten, and E. Sax. "Integration of Attribute-based Access Control into Automotive Architectures". In: *2019 IEEE Intelligent Vehicles Symposium (IV)*. June 2019, pp. 1916–1922. DOI: 10.1109/IVS.2019.8814265.

[Ryu+03] Tatyana Ryutov, Clifford Neuman, Kim Dongho, and Zhou Li. "Integrated access control and intrusion detection for web servers". In: *IEEE transactions on parallel and distributed systems* 14.9 (2003), pp. 841–850.

[Sau14] Mark Sauerwald. "CAN bus Ethernet or FPD-Link: Which is best for automotive communications". In: *Analog Appl. J* (2014).

[SBW07] Natalia Stakhanova, Samik Basu, and Johnny Wong. "A taxonomy of intrusion response systems". In: *International Journal of Information and Computer Security* 1.1-2 (2007), pp. 169–184.

[Sch11] Oliver Scheickl. "Timing constraints in distributed development of automotive real-time systems". PhD thesis. 2011.

[Sch12] Hendrik Schweppe. "Security and privacy in automotive on-board networks". PhD thesis. Télécom ParisTech, 2012.

[Sch+17] Johannes Schlatow, Marcus Nolte, Mischa Möstl, Inga Jatzkowski, Rolf Ernst, and Markus Maurer. "Towards model-based integration of component-based automotive software systems". In: *Annual Conference of the IEEE Industrial Electronics Society (IECON17)*. Beijing, China, 2017.

[Sch99] Bruce Schneier. "Attack Trees - Modeling security threats". In: *Dr. Dobb's Journal* (Dec. 1999).

[SDK19] Florian Sommer, Jürgen Dürrwang, and Reiner Kriesten. "Survey and Classification of Automotive Security Attacks". In: *Information* 10.4 (2019), p. 148.

[Sha+04] Lui Sha, Tarek Abdelzaher, Karl-Erik Årzén, Anton Cervin, Theodore Baker, Alan Burns, Giorgio Buttazzo, Marco Caccamo, John Lehoczky, and Aloysius K Mok. "Real time scheduling theory: A historical perspective". In: *Real-time systems* 28.2-3 (2004), pp. 101–155.

[SHA15] SHARCS. *Secure Hardware-Software Architectures for Robust Computing Systems (SHARCS)*. 2015.

[Shi07a] Robert W. Shirey. *Internet Security Glossary, Version 2*. RFC 4949. Aug. 2007. DOI: 10.17487/RFC4949. URL: https://rfc-editor.org/rfc/rfc4949.txt.

[Shi07b] Robert W. Shirey. *Internet Security Glossary, Version 2*. RFC 4949. Aug. 2007. DOI: 10.17487/RFC4949. URL: https://rfc-editor.org/rfc/rfc4949.txt.

[Shi+17] Hocheol Shin, Dohyun Kim, Yujin Kwon, and Yongdae Kim. "Illusion and dazzle: Adversarial optical channel exploits against lidars for automotive applications". In: *International Conference on Cryptographic Hardware and Embedded Systems*. Springer. 2017, pp. 445–467.

[Sho08] Adam Shostack. "Experiences Threat Modeling at Microsoft." In: *MODSEC@ MoDELS*. Sept. 2008.

[Sho14] Adam Shostack. *Threat Modeling: Designing for Security*. 2014.

[Sid+17] Ali Shuja Siddiqui, Yutian Gui, Jim Plusquellic, and Fareena Saqib. "Secure communication over CANBus". In: *2017 IEEE 60th International Midwest Symposium on Circuits and Systems (MWSCAS)*. Aug. 2017, pp. 1264–1267. DOI: 10.1109/MWSCAS.2017.8053160.

[SK05] Karen Seo and Stephen Kent. *Security Architecture for the Internet Protocol*. RFC 4301. Dec. 2005. DOI: 10.17487/RFC4301. URL: https://rfc-editor.org/rfc/rfc4301.txt.

[SK14] Ehsan Saeedi and Yinan Kong. "Side-channel vulnerabilities of automobiles". In: *Transaction on IoT and Cloud Computing* 2.2 (2014), pp. 1–8.

[SKK16] Hyun Min Song, Ha Rang Kim, and Huy Kang Kim. "Intrusion detection system based on the analysis of time intervals of CAN messages for in-vehicle network". In: *2016 international conference on information networking (ICOIN)*. IEEE. 2016, pp. 63–68.

[Sky] Skybox™ Security. Threat–centric vulnerability management (TCVM). https://msdn.microsoft.com/en-us/library/ee823878(v=cs.20).aspx [last access on Feb 2019].

[SM07] Karen A Scarfone and Peter M Mell. *Guide to Intrusion Detection and Prevention Systems (IDPS)| NIST*. Tech. rep. 2007.

[Sna+97] Steven R Snapp, James Brentano, Gihan V Dias, Terrance L Goan, L Todd Heberlein, Che-Lin Ho, Karl N Levitt, Biswanath Mukherjee, Stephen E Smaha, Tim Grance, et al. "DIDS (distributed intrusion detection system)-motivation, architecture, and an early prototype". In: *Internet besieged*. ACM Press/Addison-Wesley Publishing Co. 1997, pp. 211–227.

[SR90] John A Stankovic and Krithi Ramamritham. *What is predictability for real-time systems?* 1990.

[SS+12] Alireza Shameli-Sendi, Naser Ezzati-Jivan, Masoume Jabbarifar, and Michel Dagenais. "Intrusion response systems: survey and taxonomy". In: *Int. J. Comput. Sci. Netw. Secur* 12.1 (2012), pp. 1–14.

[Sta18] Statista. *Estimated amount of new cars equipped with Connected Hardware in Germany from 2017 to 2023 by segment (in million)*. https://www.statista.com/statistics/885797/new-cars-connected-hardware-in-germany, (last visited July 30, 2019). 2018.

[Ste+01] Dan Sterne, Kelly Djahandari, Brett Wilson, Bill Babson, Dan Schnackenberg, Harley Holliday, and Travis Reid. "Autonomic response to distributed denial of service attacks". In: *International Workshop on Recent Advances in Intrusion Detection*. Springer. 2001, pp. 134–149.

[Ste08] Wilfried Steiner. "TTEthernet specification". In: *TTTech Computertechnik AG, Nov* 39 (2008), p. 40.

[Ste+12] Till Steinbach, Hyung-Taek Lim, Franz Korf, Thomas C Schmidt, Daniel Herrscher, and Adam Wolisz. "Tomorrow's in-car interconnect? A competitive evaluation of IEEE 802.1 AVB and Time-Triggered Ethernet (AS6802)". In: *2012 IEEE Vehicular Technology Conference (VTC Fall)*. IEEE. 2012, pp. 1–5.

[Ste+18] Rock Stevens, Daniel Votipka, Elissa M. Redmiles, Colin Ahern, Patrick Sweeney, and Michelle L. Mazurek. "The Battle for New York: A Case Study of Applied Digital Threat Modeling at the Enterprise Level". In: *27th USENIX Security Symposium (USENIX Security 18)*. Baltimore, MD: USENIX Association, 2018, pp. 621–637. ISBN: 978-1-931971-46-1.

[Str+09] Chris Strasburg, Natalia Stakhanova, Samik Basu, and Johnny S Wong. "A framework for cost sensitive assessment of intrusion response selection". In: *Computer Software and Applications Conference, 2009. COMPSAC'09. 33rd Annual IEEE International*. Vol. 1. IEEE. 2009, pp. 355–360.

[Stu+09] Frederic Stumpf, Christian Meves, Benjamin Weyl, and Marko Wolf. "A security architecture for multipurpose ECUs in vehicles". In: *25th Joint VDI/VW Automotive Security Conference*. 2009.

[Stu+13] Ivan Studnia, Vincent Nicomette, Eric Alata, Yves Deswarte, Mohamed Kaâniche, and Youssef Laarouchi. "Survey on security threats and protection mechanisms in embedded automotive networks". In: *2013 43rd Annual IEEE/IFIP Conference on Dependable Systems and Networks Workshop (DSN-W)*. IEEE. 2013, pp. 1–12.

[Syn18] Synopsys and SAE International. *Securing the Modern Vehicle: A Study of Automotive Industry Cybersecurity Practices*. Ponemon Institute, 2018.

[SZ18] Soheil Samii and Helge Zinner. "Level 5 by layer 2: Time-sensitive networking for autonomous vehicles". In: *IEEE Communications Standards Magazine* 2.2 (2018), pp. 62–68.

[Tak+17] Junko Takahashi, Yosuke Aragane, Toshiyuki Miyazawa, Hitoshi Fuji, Hirofumi Yamashita, Keita Hayakawa, Shintarou Ukai, and Hiroshi Hayakawa. "Automotive attacks and countermeasures on LIN-bus". In: *Journal of Information Processing* 25 (2017), pp. 220–228.

[TBS18] Andrew Tomlinson, Jeremy Bryans, and Siraj Ahmed Shaikh. "Towards viable intrusion detection methods for the automotive controller area network". In: *2nd ACM Computer Science in Cars Symposium*. 2018.

[TGK00] Theodore Tryfonas, Dimitris Gritzalis, and Spyros Kokolakis. "A qualitative approach to information availability". In: *IFIP International Information Security Conference*. Springer. 2000, pp. 37–47.

[Tho15] Cadie Thompson. *A hacker figured out a way to almost completely control GM cars with OnStar*. `https://www.businessinsider.com/hackers-device-can-take-over-gm-cars-with-onstar-system-2015-7?IR=T`. July 2015.

[Thy+10] Judith Thyssen, Daniel Ratiu, Wolfgang Schwitzer, Alexander Harhurin, Martin Feilkas, and Eike Thaden. "A system for seamless abstraction layers for model-based development of embedded software". In: *Software Engineering 2010 – Workshopband (inkl. Doktorandensymposium)*. Ed. by Gregor Engels, Markus Luckey, Alexander Pretschner, and Ralf Reussner. Bonn: Gesellschaft für Informatik e.V., 2010, pp. 137–147.

[TJL15] Adrian Taylor, Nathalie Japkowicz, and Sylvain Leblanc. "Frequency-based anomaly detection for the automotive CAN bus". In: *2015 World Congress on Industrial Control Systems Security (WCICSS)*. IEEE. 2015, pp. 45–49.

[TK02] Thomas Toth and Christopher Kruegel. "Evaluating the impact of automated intrusion response mechanisms". In: *18th Annual Computer Security Applications Conference, 2002. Proceedings.* IEEE. 2002, pp. 301–310.

[Tuo+15] Shane Tuohy, Martin Glavin, Ciarán Hughes, Edward Jones, Mohan Trivedi, and Liam Kilmartin. "Intra-vehicle networks: A review". In: *IEEE Transactions on Intelligent Transportation Systems* 16.2 (2015), pp. 534–545.

[Tur02] J Turley. *Embedded processors*. Jan. 2002. URL: `Extremetech.com`.

[Ued+15] Hiroshi Ueda, Ryo Kurachi, Hiroaki Takada, Tomohiro Mizutani, Masayuki Inoue, and Satoshi Horihata. "Security authentication system for in-vehicle network". In: *SEI technical review* 81 (2015), pp. 5–9.

[Vau15] Adam Vaughan. "Adaptive Machine Learning for Modeling and Control of Non-Stationary, Near Chaotic Combustion in Real-Time". PhD thesis. Stony Brook University, 2015.

[VCHP84] Carl Von Clausewitz, Michael Eliot Howard, and Peter Paret. *On War*. Princeton University Press, 1984.

[VEV18] Marcus Vöelp and Paulo Esteves-Verissimo. "Intrusion-Tolerant Autonomous Driving". In: *2018 IEEE 21st International Symposium on Real-Time Distributed Computing (ISORC)*. May 2018, pp. 130–133. DOI: 10.1109/ISORC.2018.00026.

[VGE15] Roel Verdult, Flavio D Garcia, and Baris Ege. "Dismantling megamos crypto: Wirelessly lockpicking a vehicle Immobilizer". In: *Supplement to the 22nd USENIX Security Symposium (USENIX Security 13)*. 2015.

[VH17] Christian Vaas and Jassim Happa. "Detecting disguised processes using application-behavior profiling". In: *2017 IEEE International Symposium on Technologies for Homeland Security (HST)*. Apr. 2017, pp. 1–6.

[WA15] Armin Wasicek and Weimerskirch Andre. "Recognizing Manipulated Electronic Control Units". In: *SAE 2015 World Congress & Exhibition*. Apr. 2015.

[Wen+05] Ingomar Wenzel, Raimund Kirner, Bernhard Rieder, and Peter Puschner. "Measurement-based worst-case execution time analysis". In: *Software Technologies for Future Embedded and Ubiquitous Systems, 2005. SEUS 2005. Third IEEE Workshop on*. IEEE. 2005, pp. 7–10.

[WG11] Marko Wolf and Timo Gendrullis. "Design, implementation, and evaluation of a vehicular hardware security module". In: *International Conference on Information Security and Cryptology*. Springer. 2011, pp. 302–318.

[Wil+10] Reinhard Wilhelm, Sebastian Altmeyer, Claire Burguière, Daniel Grund, Jörg Herter, Jan Reineke, Björn Wachter, and Stephan Wilhelm. "Static timing analysis for hard real-time systems". In: *International Workshop on Verification, Model Checking, and Abstract Interpretation*. Springer. 2010, pp. 3–22.

[Win17] Stijn Winsen. "Threat modelling for future vehicles: on identifying and analysing threats for future autonomous and connected vehicles". MA thesis. University of Twente, 2017.

[WJL15] Samuel Woo, Hyo Jin Jo, and Dong Hoon Lee. "A practical wireless attack on the connected car and security protocol for in-vehicle CAN". In: *IEEE Transactions on intelligent transportation systems* 16.2 (2015), pp. 993–1006.

[WM01] Joachim Wegener and Frank Mueller. "A comparison of static analysis and evolutionary testing for the verification of timing constraints". In: *Real-time systems* 21.3 (2001), pp. 241–268.

[Woo+16] Samuel Woo, Hyo Jin Jo, In Seok Kim, and Dong Hoon Lee. "A practical security architecture for in-vehicle CAN-FD". In: *IEEE Transactions on Intelligent Transportation Systems* 17.8 (2016), pp. 2248–2261.

[Wot05] Brian Wotring. *Host integrity monitoring using Osiris and Samhain*. Elsevier, 2005.

[Wu+19] Wufei Wu, Renfa Li, Guoqi Xie, Jiyao An, Yang Bai, Jia Zhou, and Keqin Li. "A Survey of Intrusion Detection for In-Vehicle Networks". In: *IEEE Transactions on Intelligent Transportation Systems* (2019), pp. 1–15. ISSN: 1524-9050. DOI: 10.1109/TITS.2019.2908074.

[WW15] Armin Wasicek and Andre Weimerskirch. *Recognizing manipulated electronic control units*. Tech. rep. SAE Technical Paper, 2015.

[WWP04] Marko Wolf, André Weimerskirch, and Christof Paar. "Security in automotive bus systems". In: *Workshop on Embedded Security in Cars (ESCAR)*. 2004.

[WWP06] Marko Wolf, André Weimerskirch, and Christof Paar. "Secure in-vehicle communication". In: *Embedded Security in Cars*. Springer, 2006, pp. 95–109.

[Xac] *eXtensible Access Control Markup Language(XACML)*. URL: http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.html.

[Yon14] Domenick Yoney. *Tesla Model S owners hack their cars, find Ubuntu Tesla not amused*. https://www.autoblog.com/2014/04/12/tesla-model-s-owners-hack-their-cars-find-ubuntu/. Apr. 2014.

[Yoo+17] Man-Ki Yoon, Sibin Mohan, Jaesik Choi, Mihai Christodorescu, and Lui Sha. "Learning execution contexts from system call distribution for anomaly detection in smart embedded system". In: *Proceedings of the Second International Conference on Internet-of-Things Design and Implementation*. ACM. 2017, pp. 191–196.

[You+19] Clinton Young, Habeeb Olufowobi, Gedare Bloom, and Joseph Zambreno. "Automotive Intrusion Detection Based on Constant CAN Message Frequencies Across Vehicle Driving Modes". In: *ACM Workshop on Automotive Cybersecurity (AutoSec '19)*. 2019.

[Zim+10] Christopher Zimmer, Balasubramanya Bhat, Frank Mueller, and Sibin Mohan. "Time-based intrusion detection in cyber-physical systems". In: *Proceedings of the 1st ACM/IEEE International Conference on Cyber-Physical Systems*. ACM. 2010, pp. 109–118.

[ZKC16] Weiying Zeng, Mohammed AS Khalid, and Sazzadur Chowdhury. "In-Vehicle Networks Outlook: Achievements and Challenges". In: *IEEE Communications Surveys Tutorials* 18.3 (2016), pp. 1552–1571. ISSN: 1553-877X. DOI: 10.1109/COMST.2016.2521642.

[ZM14] Rafael Zalman and Albrecht Mayer. "A secure but still safe and low cost automotive communication technique". In: *Proceedings of the 51st Annual Design Automation Conference*. ACM. 2014, pp. 1–5.

[Zre+08] Saber Zrelli, Atsuko Miyaji, Yoichi Shinoda, and Thierry Ernst. "Security and access control for vehicular communications". In: *Networking and Communications, 2008. WIMOB'08. IEEE International Conference on Wireless and Mobile Computing*, IEEE. 2008, pp. 561–566.